

# **PIC BASED CONTROLLER AREA NETWORK (CAN)**

By

**AZHANI AHMAD SHAFEI**

**Final Report**

**Submitted to the Electrical & Electronics Engineering Programme  
in Partial Fulfillment of the Requirements  
for the Degree  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)**

**Universiti Teknologi Petronas  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan**

**© Copyright 2007**

**by**

**Azhani Ahmad Shafei, 2007**

# **CERTIFICATION OF APPROVAL**

## **PIC BASED CONTROLLER AREA NETWORK (CAN)**

by

Azhani Ahmad Shafei

A project dissertation submitted to the  
Electrical & Electronics Engineering Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

Approved:



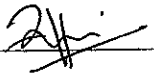
Assoc. Prof Dr. Varun Jeoti Jagadish  
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS  
TRONOH, PERAK

June 2007

## **CERTIFICATION OF ORIGINALITY**

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

  
\_\_\_\_\_  
Azhani Ahmad Shafei

## **ABSTRACT**

The field of automation and control is constantly expanding and with that the complexity in automation and control system has also been increasing. The complexity of control systems and the need to exchange data between them mean that more and more hard-wired, dedicated signal lines have to be provided. The challenges faced in automation and control system is the complexity of wiring, the complexity of the system itself and its reliability in harsh environment. Controller Area Network (CAN) provides solution in dealing with complexity of networking. CAN is also growing in popularity due to its ease of use and low costs in implementing them. CAN is as simple to use as a serial UART, and currently the cost of CAN controllers is still decreasing. The implementation of CAN not only covers the car and automation industries, but also into fields such as medical instrumentation and domestic appliances. One of the CAN microcontrollers available in the market is the Microchip's PIC18 family. The project's goal is to set up Controller Area Network (CAN) by utilizing the Microchip's PIC18. Microchip's MCP2551 CAN transceivers are used to interface the CAN controllers with the CAN bus. C language program is written to control the microcontrollers. The C program is translated to HEX file using CCS compiler. The HEX files are downloaded onto the microcontrollers. The traffic of the transmission of the messages is monitored using HyperTerminal. The microcontrollers are able to send messages among them using the CAN module. The ID of the message transmitted from one microcontroller matches the ID received from the receiving microcontroller. This work demonstrates a CAN network built using PIC microcontrollers.

## **ACKNOWLEDGEMENTS**

My deepest gratitude goes to supervisor Assoc. Prof Dr. Varun Jeoti for his useful advices and for his time spent on me for the past few months. I appreciate the opportunity he has given me to work under him for this project which has given me new insight and experiences in new area.

Special thanks to Mr Patrick Sebastian Ms. Noorashikin Binti Yahya for their kind gestures in lending me their knowledge and expertise to carry out the project. Not forgetting Ms. Nasreen Badruddin for her kind help to entertain me. Not forgetting EE FYP committee especially to Ms. Azrina and Ms. Hawa for their hospitality.

Appreciation also goes to the technicians, Miss Siti Hawa and Miss Siti Fatimah for all their help and their guidance throughout the process. Thanks to classmates for giving their view and ideas and feedbacks through some useful and enlightening discussions.

Last but not least, my love and appreciation goes to all individuals, friends and families who gave endless support and help for these past few months.

## TABLE OF CONTENTS

LIST OF FIGURES.....	ix
LIST OF TABLES .....	x
CHAPTER 1 INTRODUCTION .....	1
1.1 Background of Study.....	1
1.2 Problem Statement .....	2
1.3 Objectives and Scope of Study.....	2
1.3.1 Objectives .....	2
1.3.2 Scope of Study .....	3
CHAPTER 2 LITERATURE REVIEW .....	4
2.1 Controller Area Network.....	4
2.1.1 Example of CAN Application in industry .....	5
2.2 CAN Physical Layer.....	5
2.3 Medium Access Control.....	6
2.3.1 CSMA/CD .....	7
2.3.2 Bus Arbitration .....	8
2.3.3 Frame Formats.....	9
2.4 Microcontroller.....	10
2.4.1 PIC18F458 .....	11
2.4.2 CAN modes of operations.....	12
2.5 CAN transceivers MCP2551 .....	12
2.5.1 MCP2551 modes of operations.....	13
2.6 PIC18xxx8 CAN Functions.....	14
2.7 Wireless CAN.....	14
2.7.1 RFMAC .....	14
2.7.2 WMAC.....	15
2.7.3 On-Off Keying.....	15
CHAPTER 3 METHODOLOGY .....	16
3.1 Procedure.....	16
3.2 Tools.....	17
3.2.1 Software .....	17
3.2.2 Hardware.....	18

CHAPTER 4 RESULTS AND DISCUSSIONS .....	19
4.1 Hardware Implementation.....	19
4.1.1 Loopback mode.....	19
4.1.2 Normal mode .....	21
4.2 Communication between two microcontrollers .....	24
4.3 Communication between three microcontrollers .....	26
4.3.1 Writing C program.....	27
4.3.2 Result .....	30
CHAPTER 5 CONCLUSION & RECOMMENDATION .....	33
5.1 Conclusion.....	33
5.2 Recommendation.....	33
REFERENCES.....	34
APPENDICES.....	35
Appendix A PIC18FXX8 Data sheet .....	36
Appendix B MCP2551 Data sheet .....	37
Appendix C PIC18FXX8 CAN Driver with Prioritized Transmit Buffer .....	38
Appendix D C Code: ex_can.c.....	39
Appendix E C Code for Communication between Two Microcontrollers : Node.c.....	40
Appendix F C Code for Communication between Three Microcontrollers : StationA.c.....	41
Appendix G C Code for Communication between Three Microcontrollers : Stationb.c.....	42
Appendix H C ode for Communication between Three Microcontrollers : Stationc.c .....	43
Appendix I Hardware Connection.....	44

## LIST OF FIGURES

Figure 1 Block diagram of a AEAS-7000 with CAN interface .....	5
Figure 2 Block diagram of a CAN network .....	6
Figure 3 Examples of multiple access protocols .....	7
Figure 4 CSMA/CD Procedure [8] .....	8
Figure 5 Bit arbitration mechanism in CAN [11] .....	9
Figure 6 The CAN 2.0 B data frame .....	9
Figure 7 The PIC18458/448 Pin Diagram .....	12
Figure 8 The MCP2551 Pin Diagram .....	13
Figure 9 Modification made to the EX_CAN.c .....	19
Figure 10 Block diagram of CAN hardware for loopback mode .....	20
Figure 11 Screen capture for loopback mode traffic .....	20
Figure 12 Block diagram of CAN hardware for normal mode .....	22
Figure 13 Modification made to the EX_CAN.c for normal mode. ....	22
Figure 14 Schematic of controllers and transceivers circuit .....	23
Figure 15 Schematic of MAX232 circuit .....	23
Figure 16 Communication between the microcontrollers .....	25
Figure 17 Display at HyperTerminal at Node A .....	25
Figure 18 Communication between Node A, Node B and Node C .....	26
Figure 19 Operation between Node A, Node B and Node C .....	27
Figure 20 Flowchart for program Node A .....	28
Figure 21 Flowchart for program Node B and Node C .....	29
Figure 22 Screen capture for Node A (sending message to B) .....	30
Figure 23 Screen capture for Node B .....	31
Figure 24 Screen capture for Node C .....	31



## LIST OF TABLES

Table 1 The CAN 2.0 B data frame field name and purpose .....	10
Table 2 PIC18 Family Feature Summary.....	11
Table 3 PIC18xxx8 Function Index .....	14

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Background of Study**

The field of automation and control is constantly expanding with the introduction of innovative techniques, particularly with the advent of lower cost and higher technology tools. Day by day, the complexity in automation and control system has been increasing. For example in a car automation system, there is the need to control many electronic systems. Anti-lock Braking, Engine Management, Traction Control, Air Conditioning Control, central door locking, and powered seat and mirror controls are just some examples. The complexity of these control systems, and the need to exchange data between them meant that more and more hard-wired, dedicated signal lines had to be provided. Sensors had to be duplicated if measured parameters were needed by different controllers. Apart from the cost of the wiring looms needed to connect all these components together, the physical size of the wiring looms sometimes made it impossible to thread them around the vehicle. In addition to the cost, the increased number of connections posed serious reliability, fault diagnosis, and repair problems during both manufacture and in service.

Same can be said to automation and control in other industrial system. Some of the examples are Marine control and navigation systems, elevator control systems, agricultural machinery, production line control systems, machine tools, photo copiers, medical systems, paper making and processing machinery, packaging machinery, and textile production machinery.

All these automation and control system will benefit if there is a controller which helps reduce the complexity of wiring while reliable and is cost effective.

## **1.2 Problem Statement**

The challenges faced in automation and control system is the complexity of wiring, the complexity of the system itself and its reliability in harsh environment. CAN is growing in term of popularity for it is cost-effective and proven reliability and robustness, CAN is now also being used in many other control applications.

Controller Area Network (CAN) is inexpensive and durable. The control unit can have a single CAN interface rather than analog and digital inputs to every device in the system. Each of the devices on the network has a CAN controller chip and is therefore intelligent. All devices on the network see all transmitted messages. Each device can decide if a message is relevant or if it should be filtered. In addition, every message has a priority, so if two nodes try to send messages simultaneously, the one with the higher priority gets transmitted and the one with the lower priority gets postponed.

With this functions and features, CAN seems like a perfect candidate in automation and control system. At the moment there are few companies that produces CAN controller. One of the products available in the PIC family from Microchip includes PIC18F458, PIC18F258, PIC18F448, and PIC18F248.

## **1.3 Objectives and Scope of Study**

### ***1.3.1 Objectives***

Based on the problem discussed, the project proposes a Controller Area Network to build based on PIC microcontroller.

The objectives of this project are:

- To be able to explain the concept behind the Controller Area Network
- To be able to explain the concept of the Medium Access Control which determines the access to the transmission medium
- To be able to implement and built hardware network of Controller Area Network

using Microchip PIC microcontroller.

### ***1.3.2 Scope of Study***

The scope of study will be divided into two parts. First is the understanding and research part which includes exploration in the concept of Controller Area Network and the Medium Access Control (MAC). After that research is continued on Microchip PIC, its function and utility.

The next part is the implementation part which includes programming using C language and building the hardware and alongside testing the circuit's and network behaviour.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Controller Area Network**

Controller Area Network (CAN) is a serial bus system. The system helps communicate several intelligence devices, for example sensors and actuators. These devices or CAN nodes can try to transmit data at any point of the time [1].

CAN provide an inexpensive, durable network that helps multiple CAN devices communicate with one another. Basically, the CAN Controller implements the CAN specifications in hardware. CAN protocol is a CSMA/CD (Carrier Sense Multiple Access/Collision Detection) protocol. The protocol basically lets the nodes check the bus activity for a period before sending messages on the bus (Carrier Sense). If there is no activity, all nodes have an equal opportunity to transmit a message (Multiple Access). If two nodes start to transmit at the same time, the nodes will detect the collision and appropriate actions will be taken (Collision Detection) [1].

To solve the collision, bitwise arbitration is utilized. Logic states are defined as dominant (logic bit 0) and recessive (logic bit 1). Transmitting nodes will need to check whether the logic state out from its node appear in the bus. Dominant bit state will always win the arbitration over the Recessive bit state thus will be able to transmit its message across the bus. The lower the value in the Message Identifier, the higher the message priority. The node that lost the arbitration will have to stop transmitting its message [1].

### 2.1.1 Example of CAN Application in industry

CAN bus are first implemented in automotive applications but it has gained popularity in industrial automation, medical equipment, test equipment, and mobile machines. Other applications where CAN bus is successfully implemented are Maritime electronics, aerospace electronics, Uninterruptible Power Supply (UPS), Elevator control, Exercise equipment, and much more. [10]

One of the applications that utilizes PIC microcontroller is the implementation of CAN for AEAS-7000. The AEAS-7000 is a 16-bit gray code Absolute Encoder backbone with SSI (Synchronous-Serial-Interface) communication interface. [10] There are other microcontroller which has CAN feature in the market. One of them is CC01 from Phycore. [9]

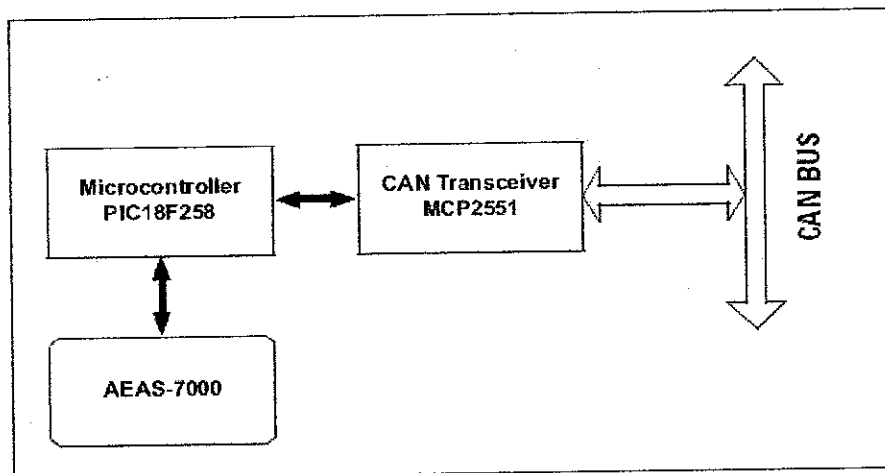


Figure 1 Block diagram of a AEAS-7000 with CAN interface

## 2.2 CAN Physical Layer

The physical layer used to implement CAN network is a differentially driven two-wire bus line with a common return. The two wires are called CAN\_H and CAN\_L. The CAN bus must be terminated at both ends by resistors with recommended value around 124 ohms [1].

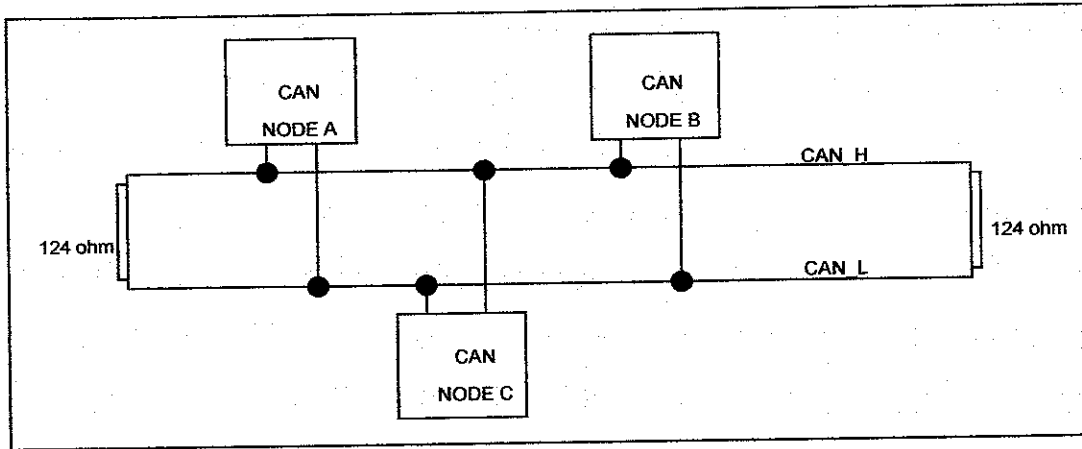


Figure 2 Block diagram of a CAN network

### 2.3 Medium Access Control

The Media Access Control (MAC) sub layer is the part of the OSI network model data link layer that determines who is allowed to access the physical media at any one time. It acts as an interface between the Logical Link Control sub layer and the network's physical layer. [1]

The Medium Access Control controls access to the transmission channel. Its purpose is to cope with problem of two or more stations sending data at the same time either by preventing the problem from happening or by recognizing the collision between data.

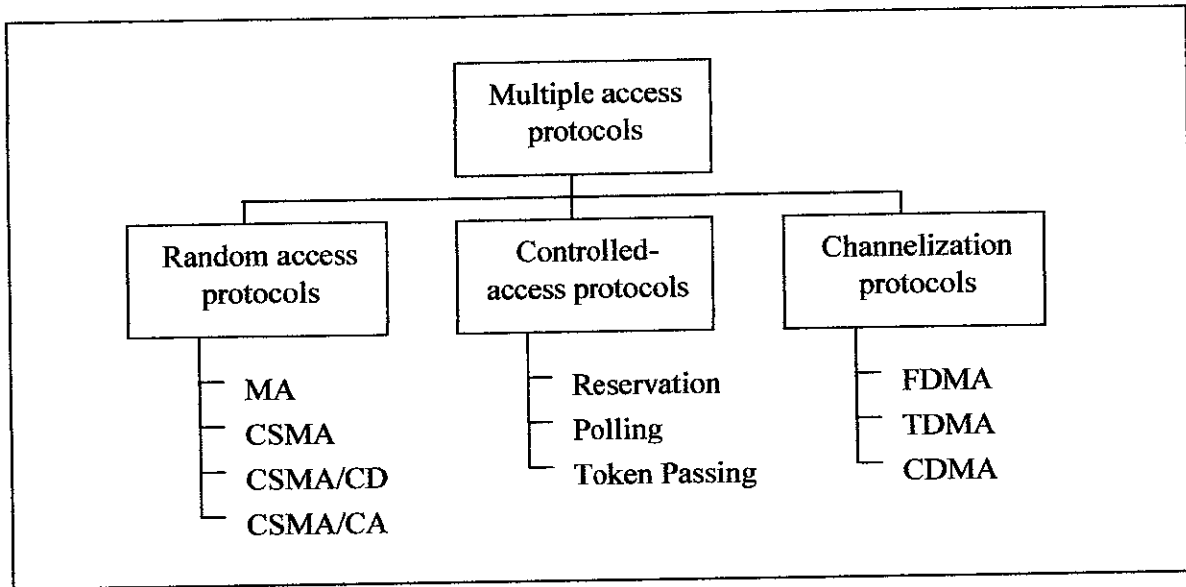


Figure 3 Examples of multiple access protocols

Many protocols have been devised to handle access to shared link. They are categorized into three groups as shown in Figure 2. Random access protocol gives the right to medium without being controlled by other stations. The problem faced is access conflict between data frames when more than one station tries to send. In controlled access, the stations consult one another to find which station is authorized to send. Channelization is the method which the available bandwidth of a link is shared in time, frequency, or through code between stations. [8]

### 2.3.1 CSMA/CD

The Medium Access Control mechanism in CAN is classified as CSMA/CD. CSMA/CD (Carrier sense multiple access with collision detection) is modification of CSMA. In this method, any station can send a frame. The station then monitors if the transmission is successful. If there is collision, the frame needs to be transmitted again. To reduce the probability of collision the second time, the station waits for an amount of time between 0 and  $2^N \times$  maximum propagation time. [8]



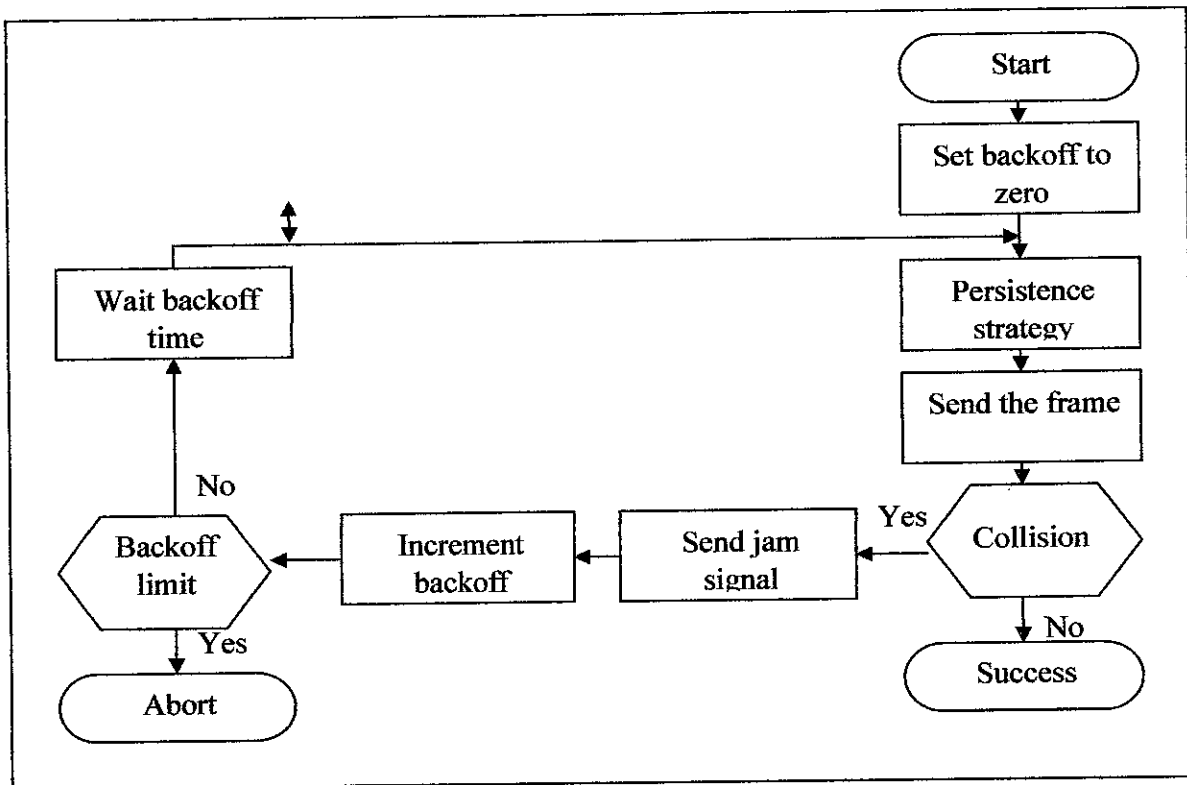


Figure 4 CSMA/CD Procedure [8]

### 2.3.2 Bus Arbitration

The identifier in a CAN message defines the priority of the message and is the basis for the medium access control for CAN. The operation is as follow.

- Station detects the bus is free for transmission. If free, then the station transmits
- During transmission, the station will keep monitoring the bus
- If, during the transmission of the Arbitration field, the node attempts to transmit a Recessive Bit but it detects Dominant Bit on the bus, it stops the transmission and wait for the bus to be free again before retransmitting data.[1]

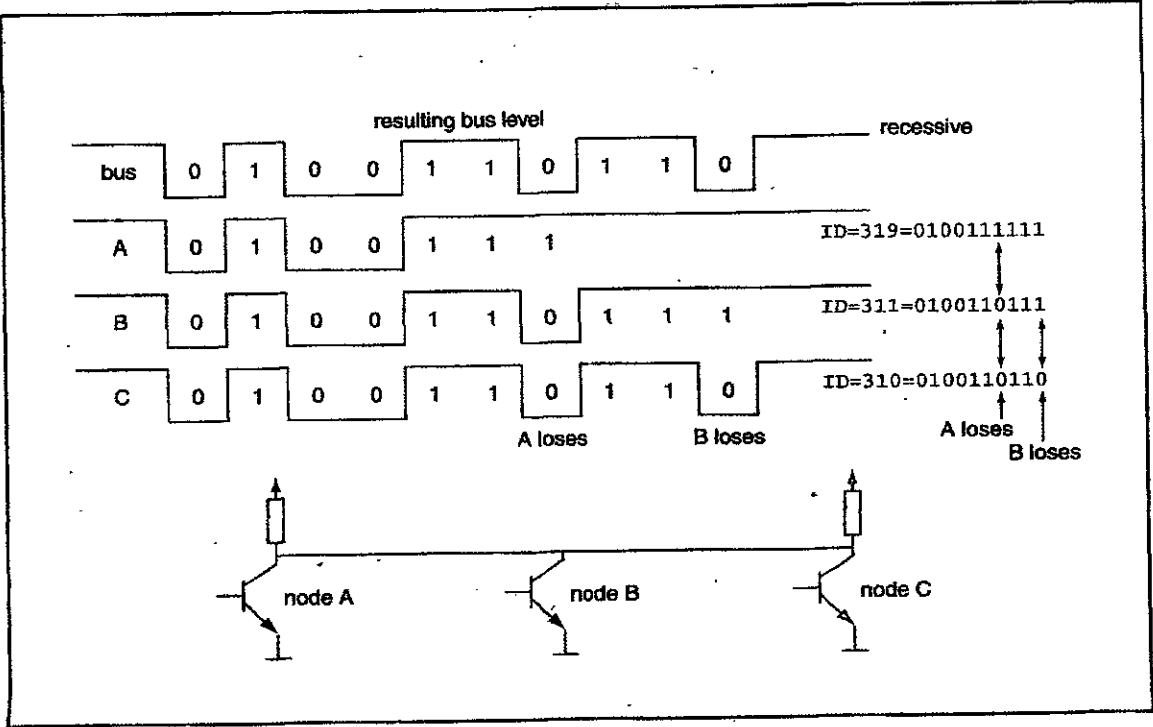


Figure 5 Bit arbitration mechanism in CAN [11]

### 2.3.3 Frame Formats

In order to perform its operation, the data are transmitted as data frames. There are two different formats for data frames, 11-bit identifier message and 29-bit identifier message. [1]

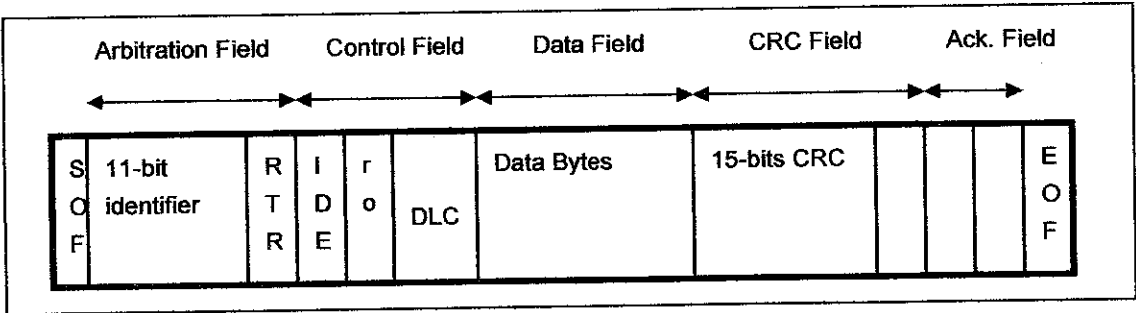


Figure 6 The CAN 2.0 B data frame

Field Name	Length (bit)	Purpose
Start of Frame (SOF)	1	Synchronization
Identifier	11	Establish message priority and identity
Remote Transmission Request (RTR)	1	If set to Dominant, frame contains data, if set to Recessive, frame is empty.
Identifier extension	1	If set to Dominant, frame is standard 11-bit identifier, if set to Recessive, frame has 29 bit identifier.
Reserved bit (r0)	2	For future use
Data Length Code (DLC)	4	Holds data byte count for message
Data field	8 bytes	Holds data
Cyclic Redundancy Check (CRC)	15	Error checking
Acknowledgement field	1	Used to ensure message has been successfully received by other nodes
End of Frame (EOF)	7	Must be recessive

Table 1 The CAN 2.0 B data frame field name and purpose

## 2.4 Microcontroller

A microcontroller is a complete computer system that consists of the processor, memory and I/O peripheral in single silicon [2]. It is capable of performing various tasks replacing the high-end microprocessor.

There are a few microcontrollers in PIC family from Microchip that has CAN solution. This PIC family from Microchip includes PIC18F458, PIC18F258, PIC18F448, and PIC18F248. The microcontroller and its features are summarized in figure below.

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	Comparators	CCP/ ECCP (PWM)	MSSP		USART	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (Bytes)	EEPROM (Bytes)					SPI	Master I2C		
PIC18F248	16K	8192	768	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F258	32K	16384	1536	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F448	16K	8192	768	256	33	8	2	1/1	Y	Y	Y	1/3
PIC18F458	32K	16384	1536	256	33	8	2	1/1	Y	Y	Y	1/3

Table 2 PIC18 Family Feature Summary

#### 2.4.1 PIC18F458

PIC18F458 has a high performance RISC (reduced instruction set computer) CPU with 33 I/O pins. It has CAN bus module features with message bit rates up to 1 Mbps. It conforms to CAN 2.0B specifications with 29-bit identifier fields, 8-byte message length, 3 transmit message buffers with prioritization, 2 receive message buffers, 6 full, 29-bit acceptance filters, prioritization of acceptance filters, multiple receive buffers for high priority messages to prevent loss due to overflow and advanced error management features [3].

The PICF458 supports communication using RS232 as a serial communication between the microcontroller and PC. This form of communication needs MAX232 as driver that acts as level voltage converter.

While having all the functions the other microcontroller has, PIC18F458 has bigger data memory and program memory size. PIC 18F458 also has more I/O channels ad more A/D channels.

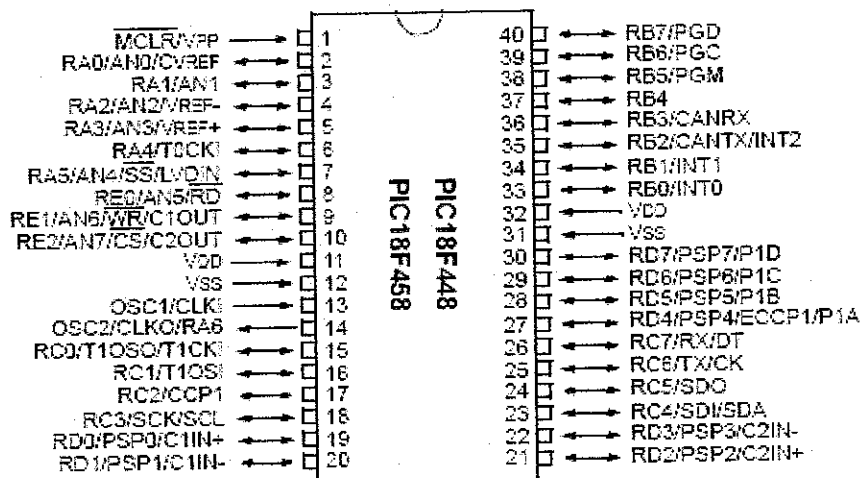


Figure 7 The PIC18458/448 Pin Diagram

#### 2.4.2 CAN modes of operations

The PIC18F458 has six main modes of CAN operation; configuration mode, disable mode, normal operation mode, listen only mode, loopback mode and error recognition mode.

When the controller is in loopback mode, the messages in its buffers will be transmitted to its receiver buffers internally without going through the CAN bus. The loopback mode is a silent mode, thus the CANTX pin will be an I/O pin. This mode is important to verify that the transceiver and receiver buffers of a controller are working correctly.

The normal mode is when the controller can actively monitors all bus messages and generates Acknowledge bits, error frames, etc. This is the standard mode for the controller. This is also the only mode in which the controller will transmit messages over the CAN bus.

#### 2.5 CAN transceivers MCP2551

The MCP2551 is a high-speed CAN, fault-tolerant device that is used as interface between a CAN protocol controller and the CAN physical bus. The MCP2551 provides differential

transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard. It is capable of operation up to 1 Mb/s. [4]

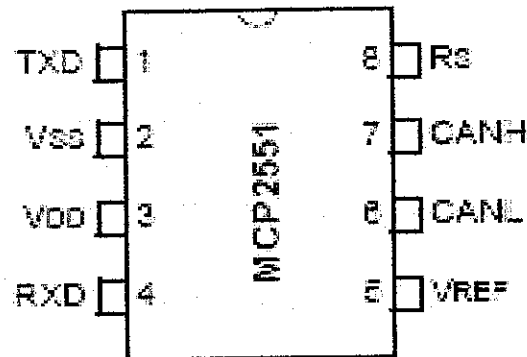


Figure 8 The MCP2551 Pin Diagram

The MCP2551 CAN stand up to 112 nodes to be connected to the physical bus. The outputs will drive a minimum load of  $45\Omega$ , (given a minimum differential input resistance of  $20\text{ k}\Omega$  and a nominal termination resistor value of  $120\Omega$ ). The MCP2551 has CANH and CANL pins that define dominant and recessive states of the nodes by the differential voltage between CANH and CANL. A dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g., 1.2V). A recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). [4]

### 2.5.1 MCP2551 modes of operations

The modes of operations of MCP2551 CAN transceiver is set by setting pin RS. There are three modes of operations which is high speed, slope control and standby. The high speed mode is set when RS pin is connected to VSS while standby mode is set by connecting RS pin to VDD.

Slope control mode is controlled by connecting an external resistor between RS and ground. Changing the value of resistor will change the slew rate.

## 2.6 PIC18xxx8 CAN Functions

All PIC18XXX8 CAN functions are grouped into the following three categories; Configuration/Initialization Functions, Module Operation Functions and Status Check Functions. These functions are used in the program to be loaded in the microcontroller.

Function	Category
CANInitialize	Configuration/Initialization
CANSetOperationMode	Configuration/Initialization
CANSetOperationModeNoWait	Configuration/Initialization
CANSetBaudRate	Configuration/Initialization
CANSetReg	Configuration/Initialization
CANSendMessage	Module Operation
CANReadMessage	Module Operation
CANAbortAll	Module Operation
CANGetTxErrorCount	Status Check
CANGetRxErrorCount	Status Check
CANIsBusOff	Status Check
CANIsTxPassive	Status Check
CANIsRxPassive	Status Check
CANIsRxReady	Status Check
CANIsTxReady	Status Check

Table 3 PIC18xxx8 Function Index

## 2.7 Wireless CAN

### 2.7.1 RFMAC

The RFMAC protocol is operated in the centralized WCAN network that consists of one master (base) node and slave nodes in the range of master node.

Remote frames are used to send periodic messages without any contention of data frames. Therefore the master node schedules all periods of data frames. If the master node wishes to have data from any node it immediately sends a remote frame to the channel. All nodes on

the network receive the remote frame and decide whether the remote frame belongs to the node by using acceptance filtering. If the remote frame identifier does not match with the acceptance filter, the slave node stays idle. A data frame is only sent when the remote frame identifier matches with the data frame identifier [5].

### **2.7.2 WMAC**

For WMAC protocol, each node must wait messages' Priority Interface Frame (PIFS) time before sending their messages. PIFS times provide message priority to each message and are derived from the scheduling method which is performed by the user application. The shortest PIFS takes the highest message priority which means shortest delay before accessing the channel. After waiting PIFS times, each node checks the channel for the second time to be sure that the channel is available for access. Hence, a message with lower PIFS will access the channel before any message with higher PIFS.

Each node has a timer called Priority Timer. The Priority Timer is set when the message is received from the channel. This prevents the nodes from the channel access during the PIFS time. This is essential since a node may wish to transmit a message during the PIFS time and sense the medium is free although there could be a node waiting its PIFS [5].

### **2.7.3 On-Off Keying**

The On-Off keying allows arbitration and acknowledgement of CAN messages. A recessive level happens when there is no signal while dominant level is sent by turning on the transmitter.

During CAN arbitration, when the level is recessive, the receiver is disabled but when in dominant state, the receiver is enabled [6].



## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Procedure**

- **Preliminary research work**

Preliminary research includes literature review on Controller Area Network. Preliminary researches are done to get as much information on the topic and to gather any data on from various sources namely reference books, thesis, the web, and experienced individual. This research work enabled student to estimate the time process and feasibility of the project to ensure success of the project.

- **Detailed analysis**

The information gathered will be learned thoroughly in order to have a firm grasp on the topic. The information gathered will be analysed so that the objectives and problem identified for the project will be further defined.

- **Microcontroller programming, hardware design and implementation**

Write or modifies existing C program in the PICC compiler. The program will be downloaded into the PIC microcontroller in the form of its HEX file. The programmer used is WARP 13.

The hardware setup includes the PIC18F458 microcontroller, MCP2551 CAN transceivers and RS232 serial communication to monitor the traffic in the PC. The project is now on this stage.

- **Testing the circuit and program**

The testing of the circuit has several parts. First is to ensure the microcontroller PIC18F458 works perfectly. Second is to ensure the RS232 hardware circuit which is used to connect the microcontroller to PC serially works. Then the circuit is tested for the communication between the nodes which includes the two microcontrollers and the two transceivers.

The programming testing is done in between the circuit testing as it used different program to test the circuits. The two microcontrollers use identical program to be downloaded onto them.

- **Final result/ hardware**

The final circuit hardware is done both on PCB board and Vera board. RS232 ports are included so that it can be connected serially to the PC via cable. The circuits are then mounted on a piece of Perspex. The PCB boards and Vera boards were held together on the Perspex using glue gun.

- **Final report and presentation**

The final report and presentation are scheduled around end of June.

## **3.2 Tools**

### **3.2.1 Software**

- **PICC compiler**

The software used to program the language for the microcontroller. The program is compiled to produce a machine-level language called the HEX file. The HEX file is to be uploaded to the microcontroller.

- **WARP 13**

The programmer used to program the microcontroller using the available HEX file.

- **Eagle**

The software used to build the schematic for the circuit.

- **Microsoft HyperTerminal**

The software used to monitor and display the data sent across serial connection to the microcontroller.

### **3.2.2 Hardware**

- **PIC18F458**

The microcontroller chosen to perform the program

- **4MHz Crystal**

Clock to feed the microcontroller

- **MCP 2551**

The transceivers for CAN communication between the microcontrollers

- **(DB 9) RS232**

Connector pin out for the serial connection

- **MAX232**

The IC driver used to adapt to RS232 signals

- **Resistors, capacitors**

- **Breadboard and Vera board**

- **Perspex as the base to hold the circuitry**

- **Glue gun/ Glue**

- **Spacer**

## CHAPTER 4

### RESULTS AND DISCUSSIONS

#### 4.1 Hardware Implementation

The hardware implementation involves several PIC18F458 communicating with each other using CAN via MCP2551 transceivers. The traffic is monitored using serial port monitor tool at PICC compiler.

##### 4.1.1 Loopback mode

This mode will allow internal transmission of messages from transmit buffer of a microcontroller to its own receive buffers without actually transmitting messages on CAN bus. [3] This mode is used to test the transmit buffer and receive buffer for each PIC18F458 microcontrollers.

Existing C program on PICC compiler, EX\_CAN.c is used to test out the CAN loopback mode. Modification is made to the EX\_CAN.c before its HEX file is downloaded to the microcontroller.

```
printf("\r\n\r\nCCS CAN EXAMPLE\r\n");

setup_timer_2(T2_DIV_BY_4,79,16); //setup up timer2 to interrupt every 1ms if using 20Mhz clock

can_init();
can_set_mode(CAN_OP_LOOPBACK); // Added this line to test for loopback

enable_interrupts(INT_TIMER2); //enable timer2 interrupt
enable_interrupts(GLOBAL); //enable all interrupts (else timer2 wont happen)

printf("\r\nRunning...");
```

Figure 9 Modification made to the EX\_CAN.c

The traffic for the transmission is monitored using RS232 with MAX232 as its driver. The traffic monitored is as expected. The messages transmitted by the microcontroller were received at the microcontroller own receive buffer thus implying its CAN module works well.

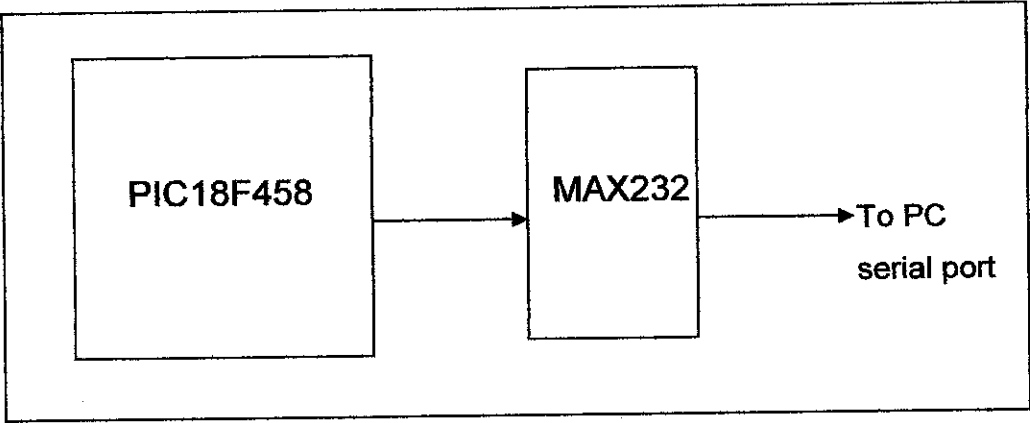


Figure 10 Block diagram of CAN hardware for loopback mode

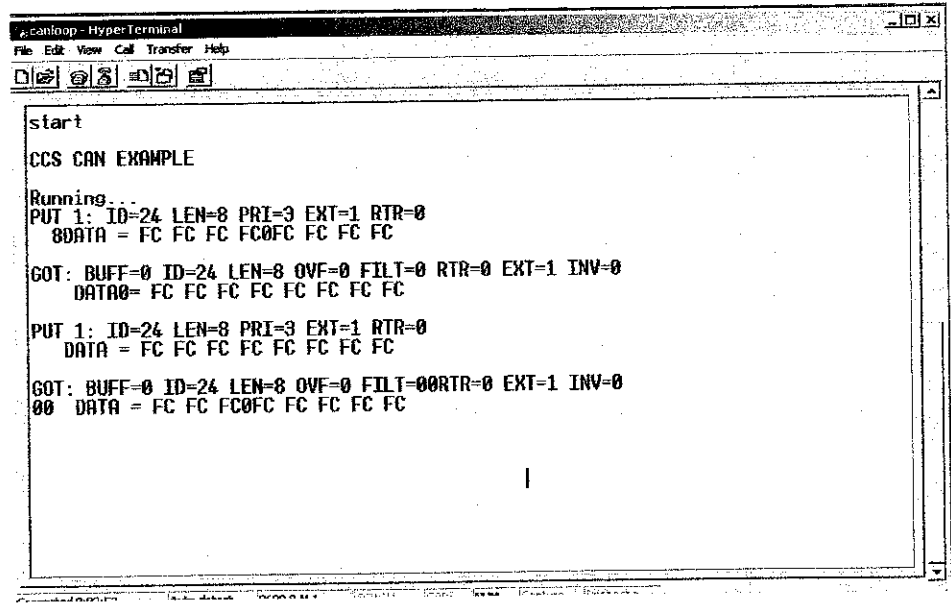


Figure 11 Screen capture for loopback mode traffic

The screen capture shows that the message from the transmit buffer managed to be sent to the receive buffer of the same PIC. The messages sent at transmit buffer and received at receive buffer has the same ID as shown above.

At the transmit buffer, LEN is the length of the data sent which is 8 byte. PRI is short for priority which in this case 3. EXT is for Identifier extension. Setting the EXT to 1 means the identifier used is the extended 29 bits identifier. RTR is for Remote Transmission Request. Setting it to 0 means the frame is the normal data frame.

At transmit buffer, BUFF=0 means the data received at buffer. OVF = 0 means there is no overflow error. INV = 0 means invalid message has not occur.

#### ***4.1.2 Normal mode***

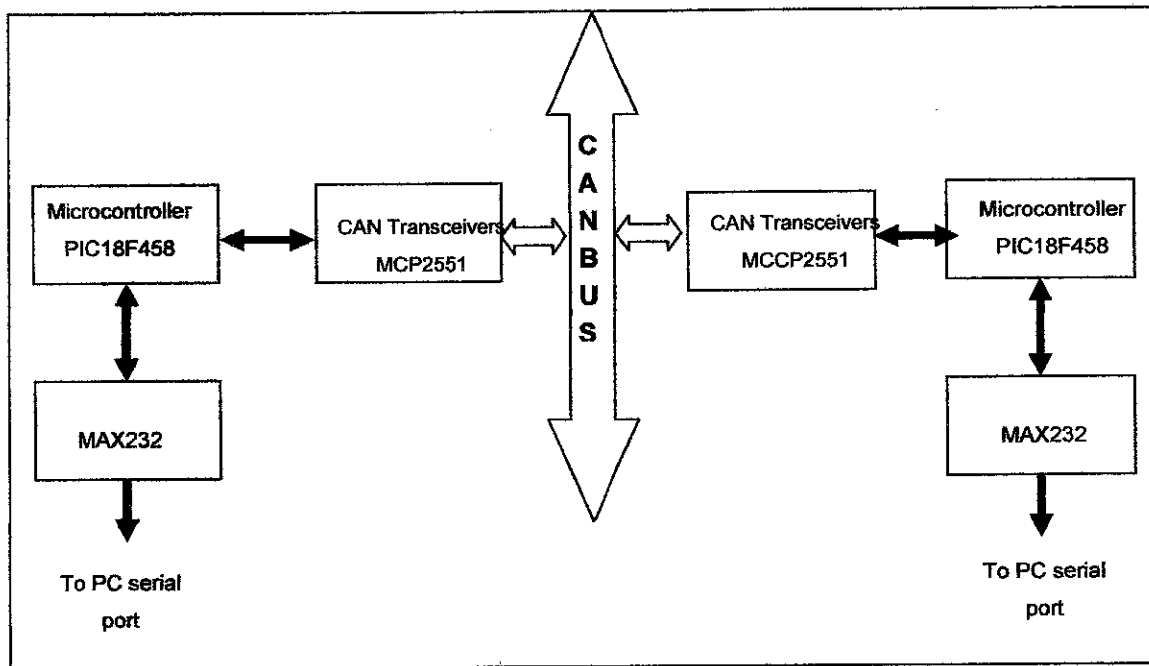
This is the standard operating mode of the PIC18F458. In this mode, the device will monitor all messages and decides whether to take the message or not. This is the only mode in which the microcontroller will transmit its messages over the CAN bus. [3]

Existing C program on PICC compiler, EX\_CAN.c is used to test out the CAN normal mode. The HEX file is downloaded into two PIC18F458 microcontrollers to allow them to communicate with each other.

CAN transceivers, MCP2551 is used as the interface between a CAN protocol controller and the physical bus. The MCP2551 provides differential transmit and receive capability for the PIC18F458 microcontroller. [3]

For the normal mode, terminating resistors of 120 ohm are used at between CANH and CANL pins. The RS pin is connected directly to ground so the transceiver is in high-speed mode.

The traffic for the transmission is monitored using RS232 with MAX232 as its driver to decide whether the messages manage to travel across the physical bus.



**Figure 12** Block diagram of CAN hardware for normal mode

```
printf("\r\n\r\nCCS CAN EXAMPLE\r\n");

setup_timer_2(T2_DIV_BY_4,79,16); //setup up timer2 to interrupt every 1ms if using 20Mhz clock

can_init();
can_set_mode(CAN_OP_NORMAL); // Added this line for normal mode

enable_interrupts(INT_TIMER2); //enable timer2 interrupt
enable_interrupts(GLOBAL); //enable all interrupts (else timer2 wont happen)

printf("\r\nRunning...");
```

**Figure 13** Modification made to the EX\_CAN.c for normal mode.

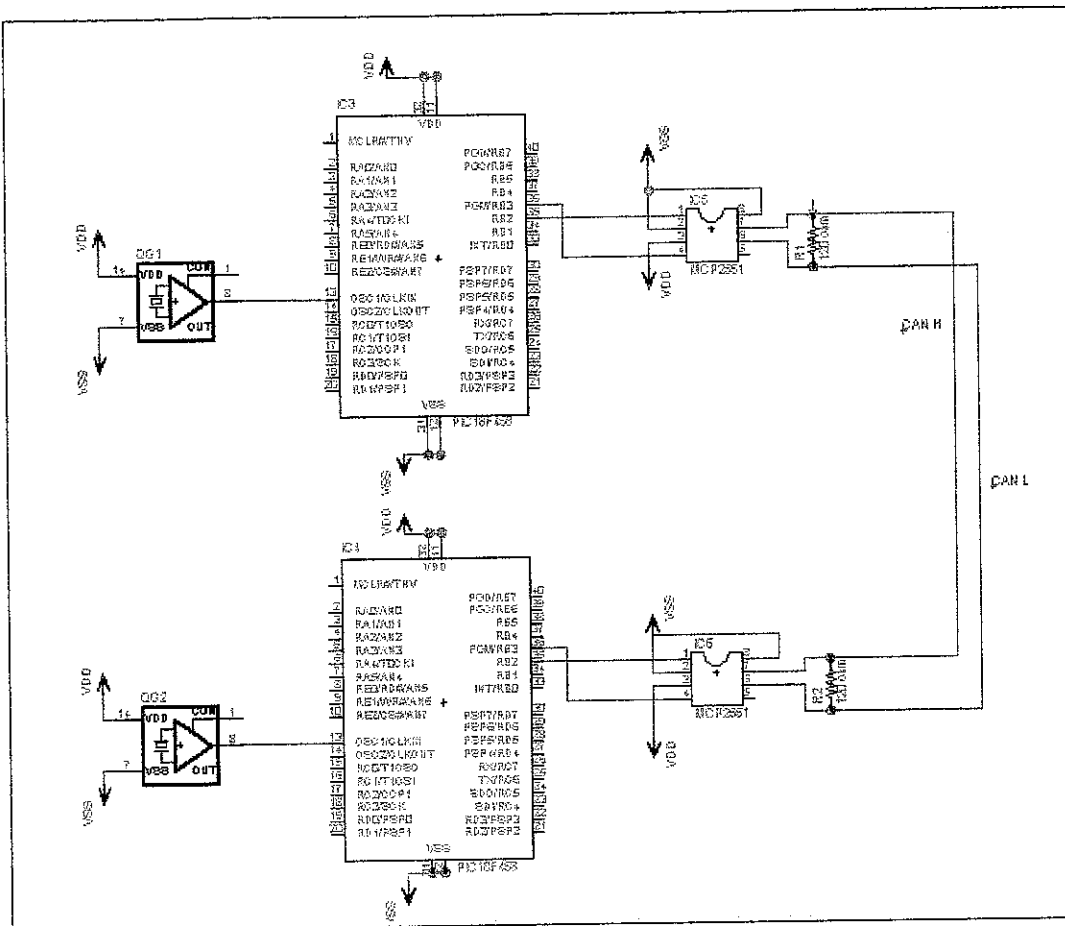


Figure 14 Schematic of controllers and transceivers circuit

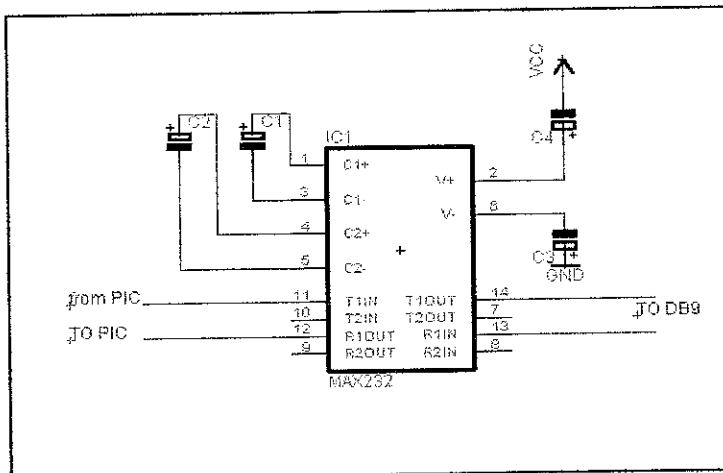


Figure 15 Schematic of MAX232 circuit



## 4.2 Communication between two microcontrollers

The two nodes have the same hardware set up which is a microcontroller, with a transceiver, and connected to an RS232 hardware circuit. The nodes have push buttons to trigger message and LEDs to indicate the messages. Both nodes have the same C program on them, Node.c as attached in the appendix.

The program written basically lets the following:

- The push button for Node 1 will trigger a sent message to Node 2.
- Node 2 will decide whether to receive the message.
- Once Node 2 accepts the message, its LEDs will light up. Then it will send back a message to Node 1 acknowledging the sent message.
- For each action, the HyperTerminal software will display the result.
- As the setup for Node 1 and Node 2 are identical, any action (sent message) from Node 2 will trigger the same response from Node 1.

From the CAN library routines, these functions are used in the program

- `can_init` - Configures the PIC18xxx8 CAN peripheral
- `can_set_baud` - Sets the baud rate control registers
- `can_set_mode` - Sets the CAN module into a specific mode
- `can_set_id` - Sets the standard and extended ID
- `can_get_id` - Gets the standard and extended ID
- `can_putd` - Sends a message/request with specified ID
- `can_getd` - Returns specific message/request and ID
- `can_kbhit` - Returns true if there is data in one of the receive buffers
- `can_tbe` - Returns true if the transmit buffer is ready to send more data

- `can_abort` - Aborts all pending transmissions

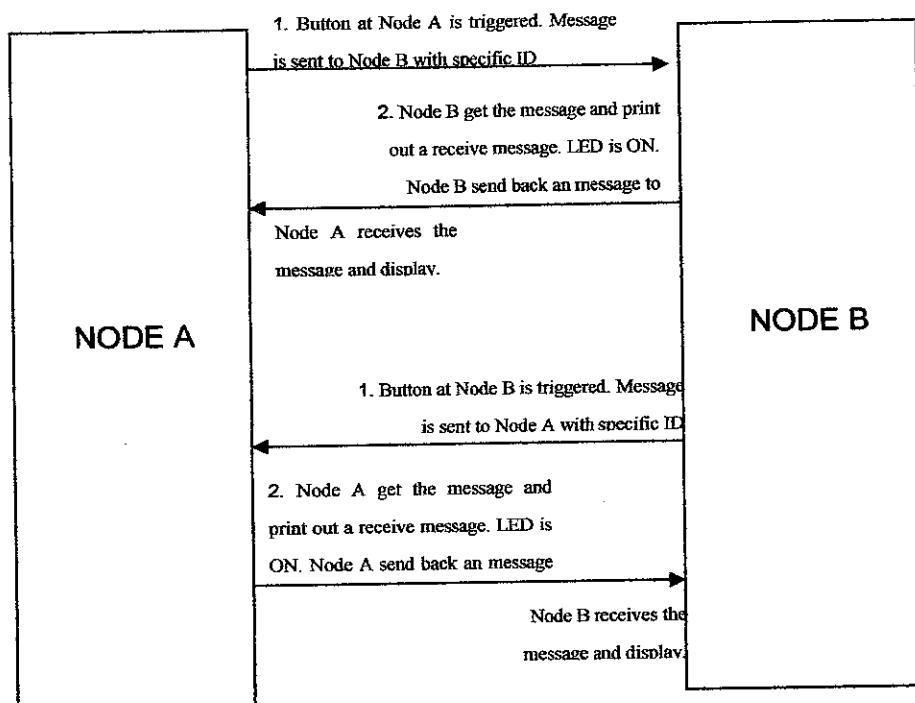


Figure 16 Communication between the microcontrollers

```

eCan - HyperTerminal
File Edit View Call Transfer Help
[Icons]
Sending message over to Node B
Sending message over to Node B
Sending message over to Node B
Sending message over to Node B
Sending message over to Node B
Sending message over to Node B
Sending message over to Node B
CCS CAN EXAMPLE
Running...
Sending message over to Node B
Sending message over to Node B
Got a message from port B
LED ON
Sending message over to Node B
Got a message from port B
LED ON
Sending message over to Node B
Got a message from port B
LED ON
Sending message over to Node B
Got a message from port B
LED ON
Sending message over to Node B
Got a message from port B
LED ON
Connected 0:01:38 Auto detect 9600 8-N-1 [SCROLL] [CAPS] [NUM] [Capture] [Print] [Echo]
  
```

Figure 17 Display at HyperTerminal at Node A

The Nodes manages to send and receive messages to each other although sometimes messages get through the channel but sometimes the message did not reach destination. But the HyperTerminal displays that the microcontrollers are able to communicate with each other and reacts to messages sent to each other.

**4.3 Communication between three microcontrollers**

The network was set with three nodes which will communicate with each other. The operation between these three nodes is shown in figure below.

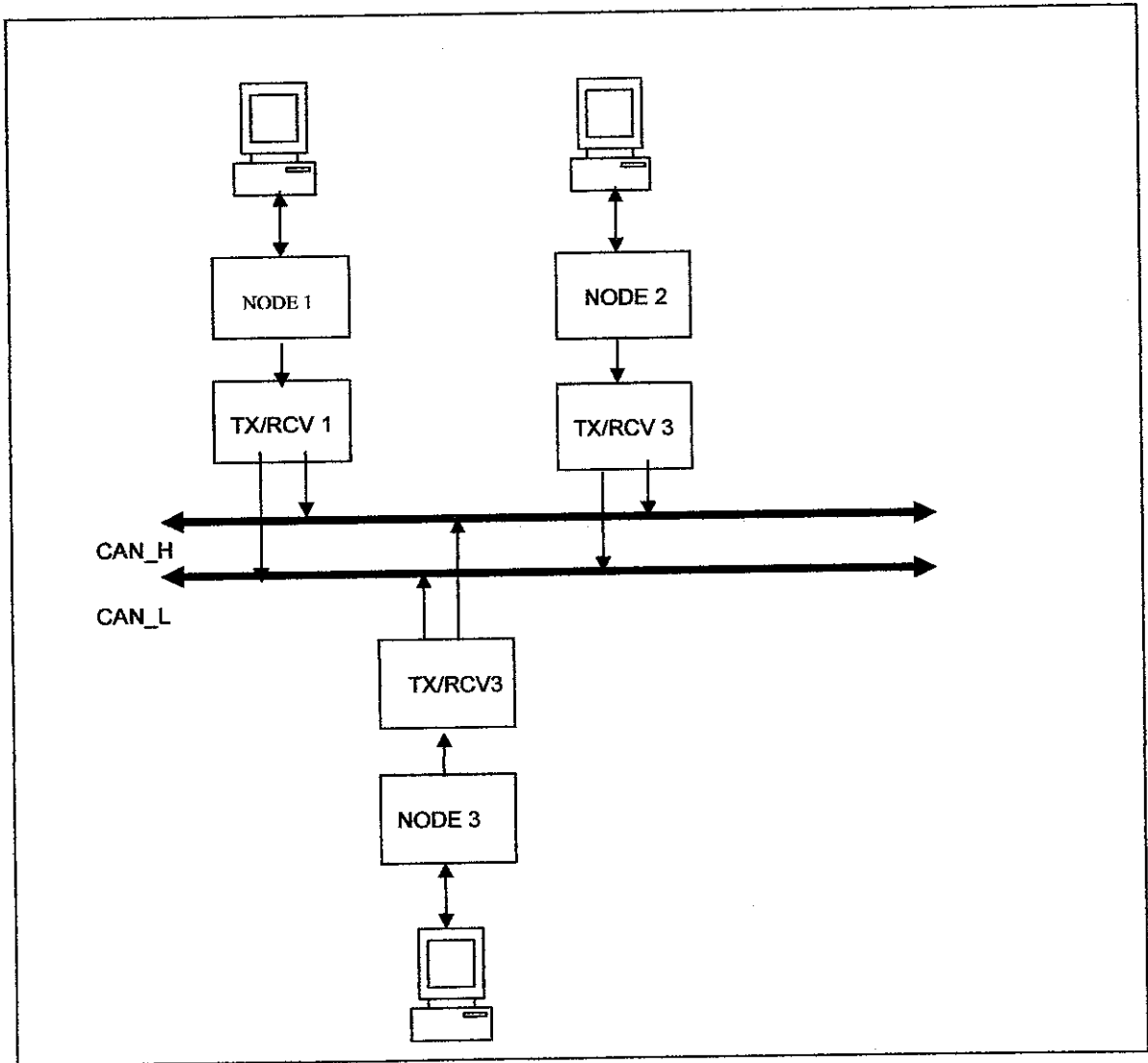


Figure 18 Communication between Node A, Node B and Node C

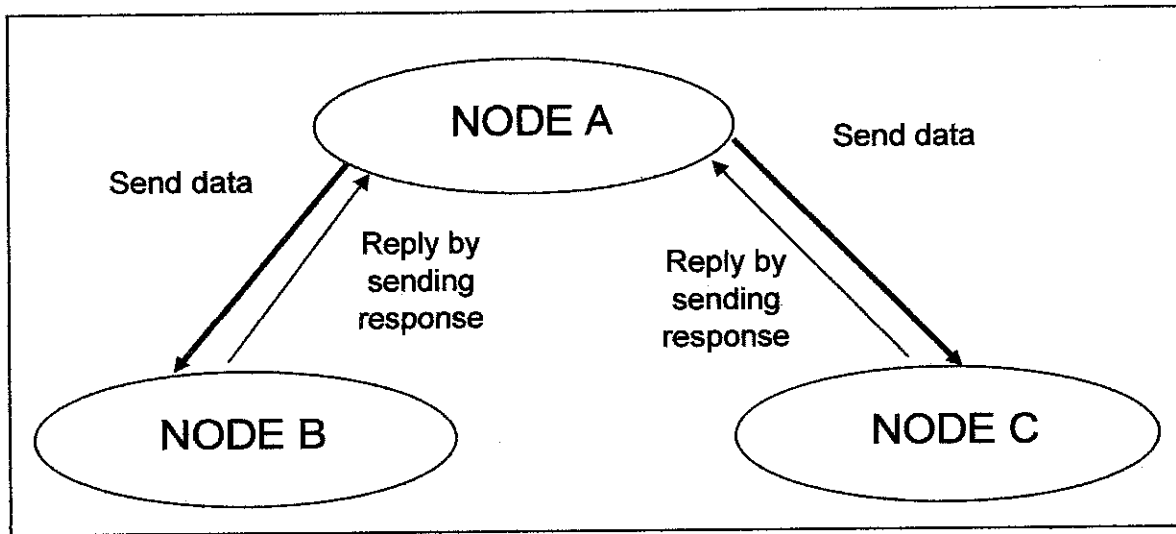


Figure 19 Operation between Node A, Node B and Node C

Node A sends data to the implied node. As CAN is a multicast protocol, the sent data will be read by both Node B and Node C. As the data is attached with its own id, Node B and Node C will compare the id and decides whether the message was intended for them. If yes, the node will reply back to Node A as a response.

#### 4.3.1 Writing C program

The C program was developed by following the sequence of the program as shown in the flow chart. The design is as follow.

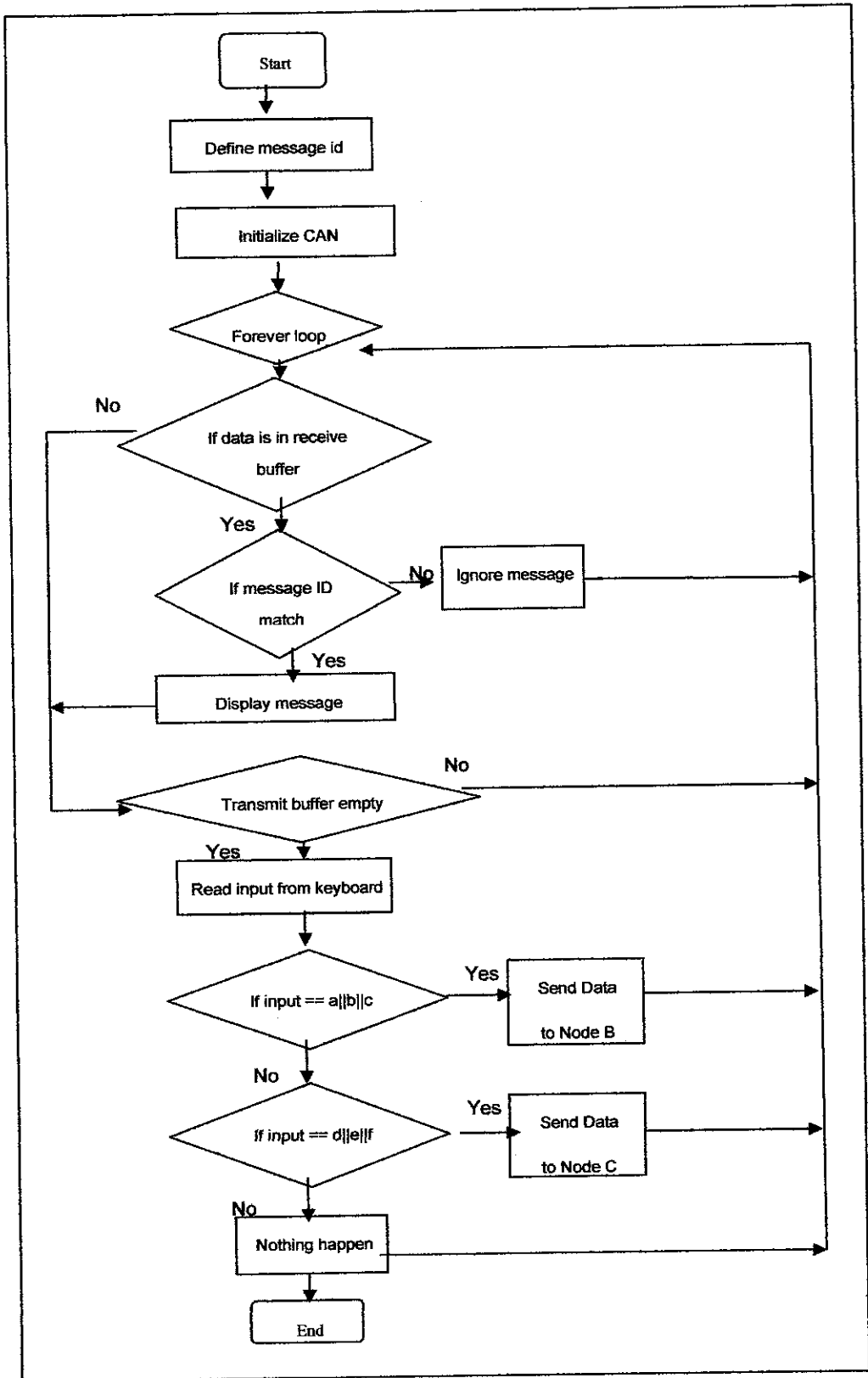


Figure 20 Flowchart for program Node A

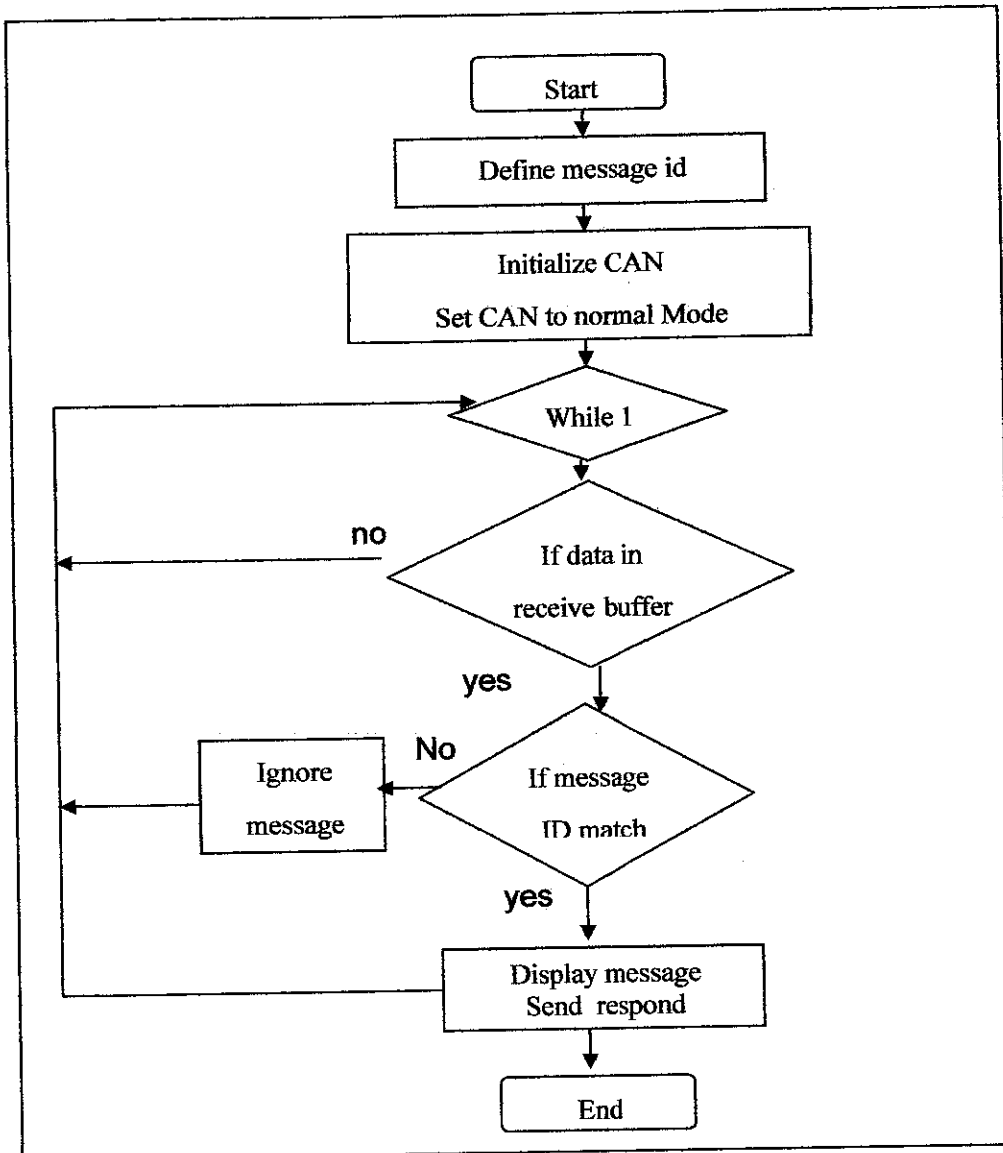


Figure 21 Flowchart for program Node B and Node C

### 4.3.2 Result

The result is monitored using Serial Input/Output Tool from PICC compiler. The screen captured is shown below.

```
\0A
Transmit buffer ready\0D
\0A
\0A
you pressed a\0D
\0A
Send data to Port B\0D
\0A
PUT 0: ID=513 LEN=8 PRI=3 EXT=0 RTR=1\0D
\0A
DATA = a a a a a a a a \0D
\0A
\0D
\0A
\0A
Data in receive buffer\0D
\0A
Message sent to Node B\0D
\0A
Transmit buffer ready\0D
\0A
\0A
you pressed c\0D
\0A
Send data to Port B\0D
\0A
PUT 0: ID=513 LEN=8 PRI=3 EXT=0 RTR=1\0D
\0A
DATA = c c c c c c c c \0D
\0A
\0D
\0A
\0A
Data in receive buffer\0D
\0A
Message sent to Node B
```

Figure 22 Screen capture for Node A (sending message to B)

```

Serial Input/Output Monitor
File Edit View Configuration Control lines Macro Manager

BARunning...\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=513 LEN=8 OUF=1 FILT=0 RTR=1 EXT=0 INU=1\BD
BA DATA = - 2 u n p R i \BD
BA\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=513 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = f n L ~ F < + \BD
BA\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=513 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=1\BD
BA DATA = " u n p R i \BD
BA\BD
BA\BD
BAMonitoring CAN... Node B\BD
BA\BD
BARunning...\BD
BAData in receive buffer\BD
BAData in receive buffer\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=513 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = t 2 n n n p R i \BD
BA\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=513 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = " ~ F < + \BD

```

Figure 23 Screen capture for Node B

```

Serial Input/Output Monitor
File Edit View Configuration Control lines Macro Manager

BA
BA\BD
BAMonitoring CAN... Node C\BD
BA\BD
BARunning...\BD
BA\BD
BAMonitoring CAN... Node C\BD
BA\BD
BARunning...\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=514 LEN=8 OUF=1 FILT=0 RTR=1 EXT=0 INU=1\BD
BA DATA = i B U D : ± r = \BD
BA\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=514 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = a + P c n S > \BD
BA\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=514 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = " \B D : ± r = \BD
BA\BD
BA\BD
BAMonitoring CAN... Node C\BD
BA\BD
BARunning...\BD
BAData in receive buffer\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=514 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = a + P c n S > \BD
BA\BD
BAData in receive buffer\BD
BAData in receive buffer\BD
BAData is from Node A\BD
BAGOT: BUFF=0 ID=514 LEN=8 OUF=0 FILT=0 RTR=1 EXT=0 INU=0\BD
BA DATA = a + P c n S > \BD

```

Figure 24 Screen capture for Node C



Comparing Figure 7 and Figure 8, it shows that the ID for data sent by Node A matches the ID for data received by Node B. The statement "Message sent to Node B" on Figure 7 shows that Node B has already responded to Node A after receiving the message.

The Statement "Data in receive buffer" which is shown repeatedly in Figure 9 occurs because the data is meant for Node B, not Node C. As CAN is a multicast network, all nodes receives all the broadcasted data in the channel.

## **CHAPTER 5**

### **CONCLUSION & RECOMMENDATION**

#### **5.1 Conclusion**

The objective of the project is to build a PIC based Controller Area Network. A network is built between two nodes which communicate with each other. Each node is controlled by a PIC microcontroller. The messages are transmitted via CAN transceivers MCP 2551 which connects the two nodes using CAN buses, CAN\_H and CAN\_L. these CAN buses need to have terminating resistor at the end of its connection. The microcontrollers are serially connected to PC via RS232. The data monitoring uses HyperTerminal as a displaying tool.

Programming plays a great part in the project. The programs for this project are written in C by utilizing the functions from CAN library. The CAN network built using PIC microcontroller is shown to be functional. The microcontrollers are able to send and receive messages, as well as requesting messages IDs among others. The microcontrollers are able to respond according to the messages sent by other microcontroller.

#### **5.2 Recommendation**

For future work on CAN, the system can have more complex network circuit which has large number of nodes with each node controlling several control applications. The system should be made robust, reliable and efficient.

For future studies, one can focus other functions in CAN module available in Microchip's PIC such as error recognition mode and listen only mode and other functions such situation such as message overflow, error handing and synchronization.

## REFERENCES

1. Farsi M., & Barbosa M. (2000) *CANopen Implementation Applications to Industrial Networks*, Research Studies Press Ltd.
2. Barnett, Cox & O'Cull (2004) *Embedded C Programming and the Microchip PIC*, McGraw-Hill
3. *PIC18FXX8 Data Sheet*, Microchip, Microchip Technology Inc. 2001
4. *MCP2551 Data Sheet*, Microchip, Microchip Technology Inc. 2001
5. A. Kutlu, H. Ekiz, E.T. Powner, *Wireless Control Area Network*, London, 1996
6. CANRF UHF Wireless CAN Module, Automation Artisan Inc. 2002
7. *PIC18FXX8 CAN Driver with Prioritized Transmit Buffer*, Microchip, Microchip Technology Inc. 2001
8. Forouzan B. A., (2004) *Data Communication Network*, McGraw-Hill, New York, USA.
9. E. Yin, M. Sibenac, B. Kirkwood, *Implementing CANopen in Autonomous Underwater Vehicles*, Stanford University 2003
10. *Controller Area Network (CAN) Bus for AEAS-7000 Application Note 5223*, Agilent Technologies, 2005
11. M. Farsi, K. Ratcliff, M. Barbosa, *An Overview of Controller Area Network*, Computing and Control Engineering Journal, 1999

## **APPENDICES**

**APPENDIX A**  
**PIC18FXX8 DATA SHEET**



# PIC18FXX8

## 28/40-Pin High-Performance, Enhanced Flash Microcontrollers with CAN

### High-Performance RISC CPU:

- Linear program memory addressing up to 2 Mbytes
- Linear data memory addressing to 4 Kbytes
- Up to 10 MIPS operation
- DC – 40 MHz clock input
- 4 MHz-10 MHz oscillator/clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 8 x 8 Single-Cycle Hardware Multiplier

### Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer2 module: 8-bit timer/counter with 8-bit period register (time base for PWM)
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option – Timer1/Timer3
- Capture/Compare/PWM (CCP) modules; CCP pins can be configured as:
  - Capture input: 16-bit, max resolution 6.25 ns
  - Compare: 16-bit, max resolution 100 ns ( $T_{CY}$ )
  - PWM output: PWM resolution is 1 to 10-bit  
Max. PWM freq. @: 8-bit resolution = 156 kHz  
10-bit resolution = 39 kHz
- Enhanced CCP module which has all the features of the standard CCP module, but also has the following features for advanced motor control:
  - 1, 2 or 4 PWM outputs
  - Selectable PWM polarity
  - Programmable PWM dead time
- Master Synchronous Serial Port (MSSP) with two modes of operation:
  - 3-wire SPI™ (Supports all 4 SPI modes)
  - I<sup>2</sup>C™ Master and Slave mode
- Addressable USART module:
  - Supports interrupt-on-address bit

### Advanced Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter module (A/D) with:
  - Conversion available during Sleep
  - Up to 8 channels available
- Analog Comparator module:
  - Programmable input and output multiplexing
- Comparator Voltage Reference module
- Programmable Low-Voltage Detection (LVD) module:
  - Supports interrupt-on-Low-Voltage Detection
- Programmable Brown-out Reset (BOR)

### CAN bus Module Features:

- Complies with ISO CAN Conformance Test
- Message bit rates up to 1 Mbps
- Conforms to CAN 2.0B Active Spec with:
  - 29-bit Identifier Fields
  - 8-byte message length
  - 3 Transmit Message Buffers with prioritization
  - 2 Receive Message Buffers
  - 6 full, 29-bit Acceptance Filters
  - Prioritization of Acceptance Filters
  - Multiple Receive Buffers for High Priority Messages to prevent loss due to overflow
  - Advanced Error Management Features

### Special Microcontroller Features:

- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator
- Programmable code protection
- Power-saving Sleep mode
- Selectable oscillator options, including:
  - 4x Phase Lock Loop (PLL) of primary oscillator
  - Secondary Oscillator (32 kHz) clock input
- In-Circuit Serial Programming™ (ICSP™) via two pins

### Flash Technology:

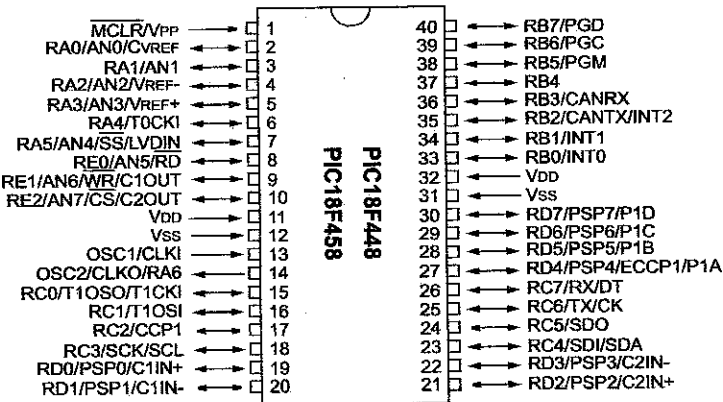
- Low-power, high-speed Enhanced Flash technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges

# PIC18FXX8

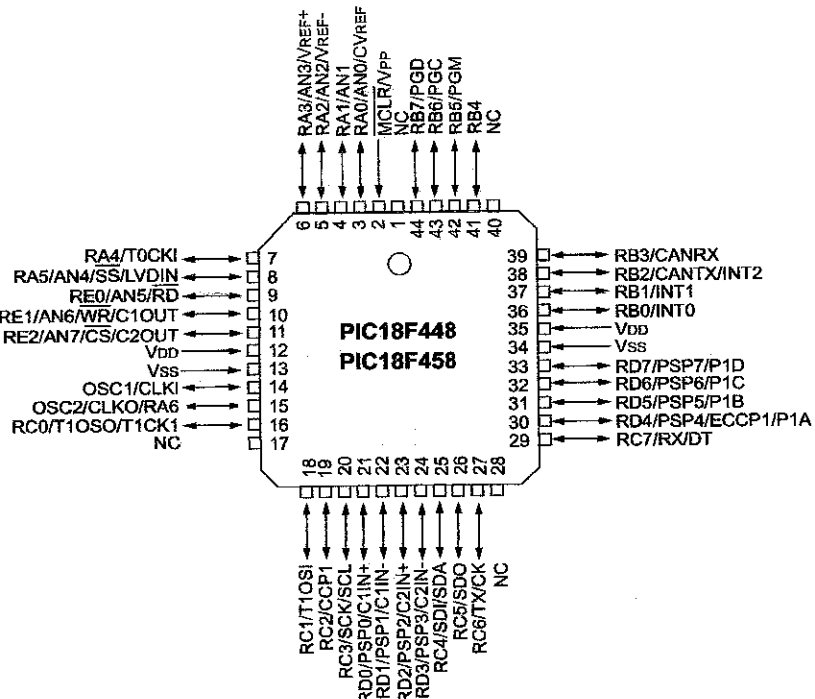
Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	Comparators	CCP/ ECCP (PWM)	MSSP		USART	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI™	Master I²C™		
PIC18F248	16K	8192	768	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F258	32K	16384	1536	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F448	16K	8192	768	256	33	8	2	1/1	Y	Y	Y	1/3
PIC18F458	32K	16384	1536	256	33	8	2	1/1	Y	Y	Y	1/3

## Pin Diagrams

PDIP

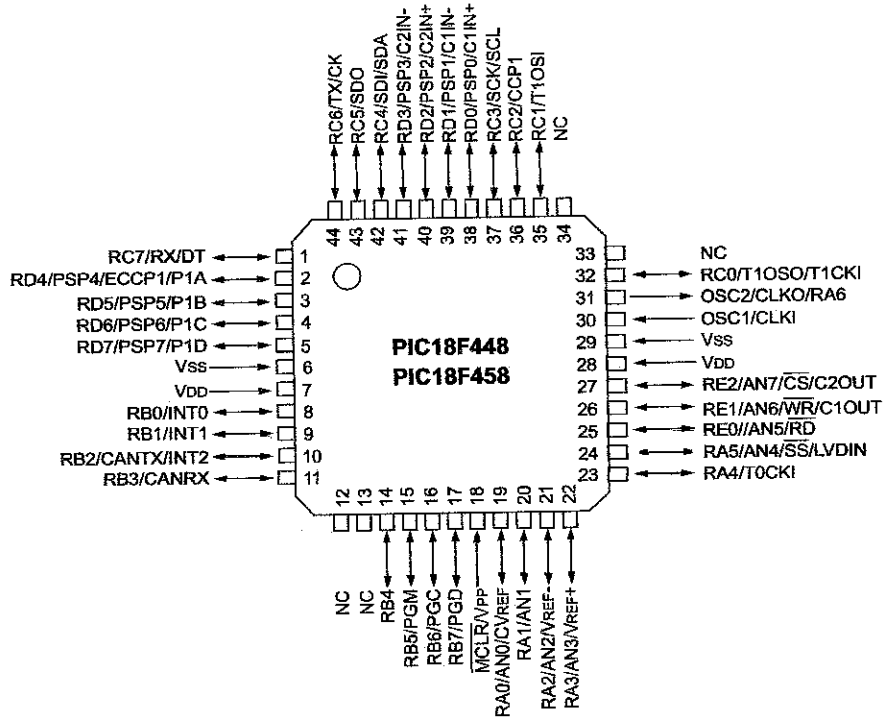


PLCC

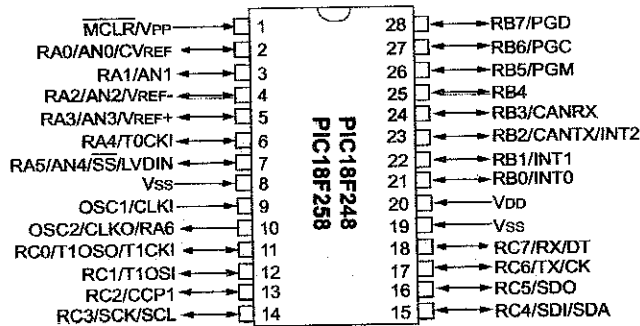


## Pin Diagrams (Continued)

### TQFP



### SPDIP, SOIC





**APPENDIX B**  
**MCP2551 DATA SHEET**

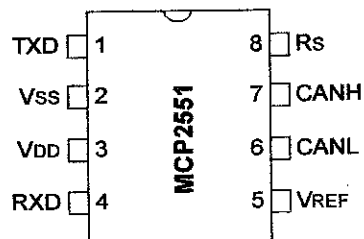
## High-Speed CAN Transceiver

### Features

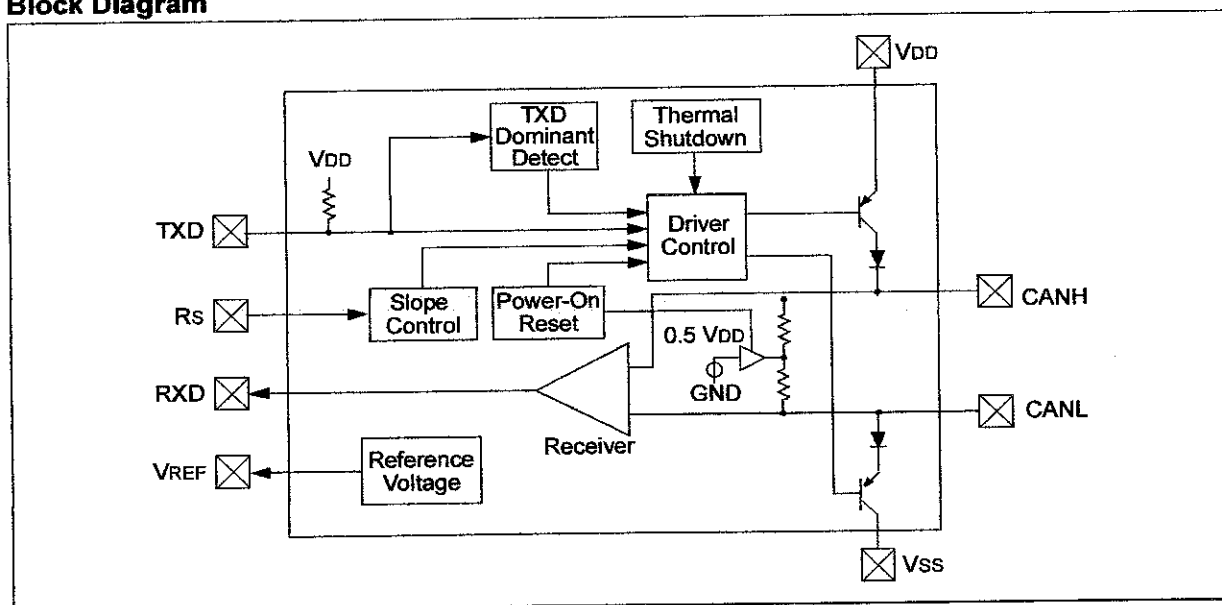
- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent dominant) on TXD input
- Power-on reset and voltage brown-out protection
- An unpowered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Protection against high-voltage transients
- Automatic thermal shutdown protection
- Up to 112 nodes can be connected
- High noise immunity due to differential bus implementation
- Temperature ranges:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

### Package Types

#### PDIP/SOIC



### Block Diagram



# MCP2551

---

NOTES:

## 1.0 DEVICE OVERVIEW

The MCP2551 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2551 provides differential transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s.

Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources (EMI, ESD, electrical transients, etc.).

### 1.1 Transmitter Function

The CAN bus has two states: Dominant and Recessive. A dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g., 1.2V). A recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). The dominant and recessive states correspond to the low and high state of the TXD input pin, respectively. However, a dominant state initiated by another CAN node will override a recessive state on the CAN bus.

#### 1.1.1 MAXIMUM NUMBER OF NODES

The MCP2551 CAN outputs will drive a minimum load of 45Ω, allowing a maximum of 112 nodes to be connected (given a minimum differential input resistance of 20 kΩ and a nominal termination resistor value of 120Ω).

### 1.2 Receiver Function

The RXD output pin reflects the differential bus voltage between CANH and CANL. The low and high states of the RXD output pin correspond to the dominant and recessive states of the CAN bus, respectively.

### 1.3 Internal Protection

CANH and CANL are protected against battery short-circuits and electrical transients that can occur on the CAN bus. This feature prevents destruction of the transmitter output stage during such a fault condition.

The device is further protected from excessive current loading by thermal shutdown circuitry that disables the output drivers when the junction temperature exceeds a nominal limit of 165°C. All other parts of the chip remain operational and the chip temperature is lowered due to the decreased power dissipation in the transmitter outputs. This protection is essential to protect against bus line short-circuit-induced damage.

## 1.4 Operating Modes

The Rs pin allows three modes of operation to be selected:

- High-Speed
- Slope-Control
- Standby

These modes are summarized in Table 1-1.

When in High-speed or Slope-control mode, the drivers for the CANH and CANL signals are internally regulated to provide controlled symmetry in order to minimize EMI emissions.

Additionally, the slope of the signal transitions on CANH and CANL can be controlled with a resistor connected from pin 8 (Rs) to ground, with the slope proportional to the current output at Rs, further reducing EMI emissions.

#### 1.4.1 HIGH-SPEED

High-speed mode is selected by connecting the Rs pin to Vss. In this mode, the transmitter output drivers have fast output rise and fall times to support high-speed CAN bus rates.

#### 1.4.2 SLOPE-CONTROL

Slope-control mode further reduces EMI by limiting the rise and fall times of CANH and CANL. The slope, or slew rate (SR), is controlled by connecting an external resistor (REXT) between Rs and VOL (usually ground). The slope is proportional to the current output at the Rs pin. Since the current is primarily determined by the slope-control resistance value REXT, a certain slew rate is achieved by applying a respective resistance. Figure 1-1 illustrates typical slew rate values as a function of the slope-control resistance value.

#### 1.4.3 STANDBY MODE

The device may be placed in standby or "SLEEP" mode by applying a high-level to Rs. In SLEEP mode, the transmitter is switched off and the receiver operates at a lower current. The receive pin on the controller side (RXD) is still functional but will operate at a slower rate. The attached microcontroller can monitor RXD for CAN bus activity and place the transceiver into normal operation via the Rs pin (at higher bus rates, the first CAN message may be lost).

# MCP2551

TABLE 1-1: MODES OF OPERATION

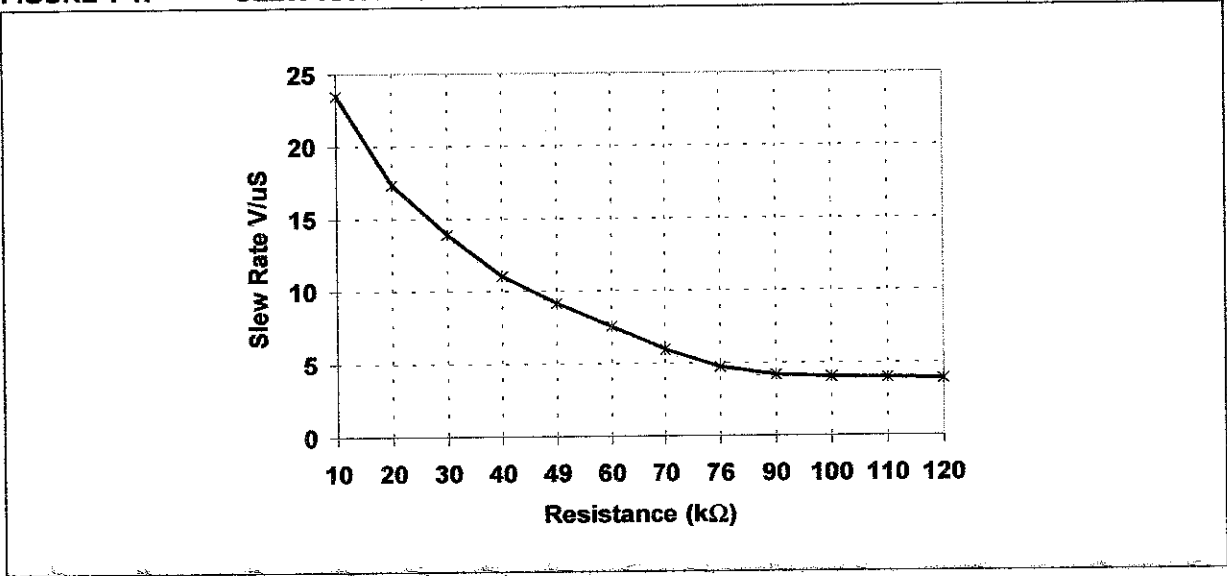
Mode	Current at R <sub>s</sub> Pin	Resulting Voltage at R <sub>s</sub> Pin
Standby	-I <sub>RS</sub> < 10 µA	V <sub>RS</sub> > 0.75 V <sub>DD</sub>
Slope-control	10 µA < -I <sub>RS</sub> < 200 µA	0.4 V <sub>DD</sub> < V <sub>RS</sub> < 0.6 V <sub>DD</sub>
High-speed	-I <sub>RS</sub> < 610 µA	0 < V <sub>RS</sub> < 0.3V <sub>DD</sub>

TABLE 1-2: TRANSCEIVER TRUTH TABLE

V <sub>DD</sub>	V <sub>RS</sub>	TXD	CANH	CANL	Bus State <sup>(1)</sup>	RxD <sup>(1)</sup>
4.5V ≤ V <sub>DD</sub> ≤ 5.5V	V <sub>RS</sub> < 0.75 V <sub>DD</sub>	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	V <sub>RS</sub> > 0.75 V <sub>DD</sub>	X	Not Driven	Not Driven	Recessive	1
V <sub>POR</sub> < V <sub>DD</sub> < 4.5V (See Note 3)	V <sub>RS</sub> < 0.75 V <sub>DD</sub>	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	V <sub>RS</sub> > 0.75 V <sub>DD</sub>	X	Not Driven	Not Driven	Recessive	1
0 < V <sub>DD</sub> < V <sub>POR</sub>	X	X	Not Driven/ No Load	Not Driven/ No Load	High Impedance	X

- Note 1:** If another bus node is transmitting a dominant bit on the CAN bus, then RxD is a logic '0'.  
**2:** X = "don't care".  
**3:** Device drivers will function, although outputs are not ensured to meet the ISO-11898 specification.

FIGURE 1-1: SLEW RATE VS. SLOPE-CONTROL RESISTANCE VALUE



## 1.5 TXD Permanent Dominant Detection

If the MCP2551 detects an extended low state on the TXD input, it will disable the CANH and CANL output drivers in order to prevent the corruption of data on the CAN bus. The drivers are disabled if TXD is low for more than 1.25 ms (minimum). This implies a maximum bit time of 62.5  $\mu$ s (16 kb/s bus rate), allowing up to 20 consecutive transmitted dominant bits during a multiple bit error and error frame scenario. The drivers remain disabled as long as TXD remains low. A rising edge on TXD will reset the timer logic and enable the CANH and CANL output drivers.

## 1.6 Power-on Reset

When the device is powered on, CANH and CANL remain in a high-impedance state until VDD reaches the voltage-level VPORH. In addition, CANH and CANL will remain in a high-impedance state if TXD is low when VDD reaches VPORH. CANH and CANL will become active only after TXD is asserted high. Once powered on, CANH and CANL will enter a high-impedance state if the voltage level at VDD falls below VPORL, providing voltage brown-out protection during normal operation.

## 1.7 Pin Descriptions

The 8-pin pinout is listed in Table 1-3.

**TABLE 1-3: MCP2551 PINOUT**

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	VSS	Ground
3	VDD	Supply Voltage
4	RXD	Receive Data Output
5	VREF	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	RS	Slope-Control Input

### 1.7.1 TRANSMITTER DATA INPUT (TXD)

TXD is a TTL-compatible input pin. The data on this pin is driven out on the CANH and CANL differential output pins. It is usually connected to the transmitter data output of the CAN controller device. When TXD is low, CANH and CANL are in the dominant state. When TXD is high, CANH and CANL are in the recessive state, provided that another CAN node is not driving the CAN bus with a dominant state. TXD has an internal pull-up resistor (nominal 25 k $\Omega$  to VDD).

### 1.7.2 GROUND SUPPLY (VSS)

Ground supply pin.

### 1.7.3 SUPPLY VOLTAGE (VDD)

Positive supply voltage pin.

### 1.7.4 RECEIVER DATA OUTPUT (RXD)

RXD is a CMOS-compatible output that drives high or low depending on the differential signals on the CANH and CANL pins and is usually connected to the receiver data input of the CAN controller device. RXD is high when the CAN bus is recessive and low in the dominant state.

### 1.7.5 REFERENCE VOLTAGE (VREF)

Reference Voltage Output (Defined as VDD/2).

### 1.7.6 CAN LOW (CANL)

The CANL output drives the low side of the CAN differential bus. This pin is also tied internally to the receive input comparator.

### 1.7.7 CAN HIGH (CANH)

The CANH output drives the high-side of the CAN differential bus. This pin is also tied internally to the receive input comparator.

### 1.7.8 SLOPE RESISTOR INPUT (RS)

The RS pin is used to select High-speed, Slope-control or Standby modes via an external biasing resistor.

## **APPENDIX C**

### **PIC18FXX8 CAN DRIVER WITH PRIORITIZED TRANSMIT BUFFER**

---

## PIC18XXX8 CAN Driver with Prioritized Transmit Buffer

---

Author: Gaurana Kavaiva

Microchip Technology Inc.

### INTRODUCTION

The Microchip PIC18XXX8 family of microcontrollers provide an integrated Controller Area Network (CAN) solution along with other PICmicro® features. Although originally intended for the automotive industry, CAN is finding its way into other control applications. In CAN, a protocol message with highest priority wins the bus arbitration and maintains the bus control. For minimum message latency and bus control, messages should be transmitted on a priority basis.

Because of the wide applicability of the CAN protocol, developers are faced with the often cumbersome task of dealing with the intricate details of CAN registers. This application note presents a software library that hides the details of CAN registers, and discusses the design of the CAN driver with prioritized Transmit buffer implementation. This software library allows developers to focus their efforts on application logic, while minimizing their interaction with CAN registers.

If the controller has heavy transmission loads, it is advisable to use software Transmit buffers to reduce message latency. Firmware also supports user defined Transmit buffer size. If the defined size of a Transmit buffer is more than that available in hardware (8), the CAN driver will use 14 bytes of general purpose RAM for each extra buffer.

For details about the PIC18 family of microcontrollers, refer to the PIC18CXX8 Data Sheet (DS30475), the PIC18FXX8 Data Sheet (DS41159), and the PICmicro® 18C MCU Family Reference Manual (DS39500).

### CAN MODULE OVERVIEW

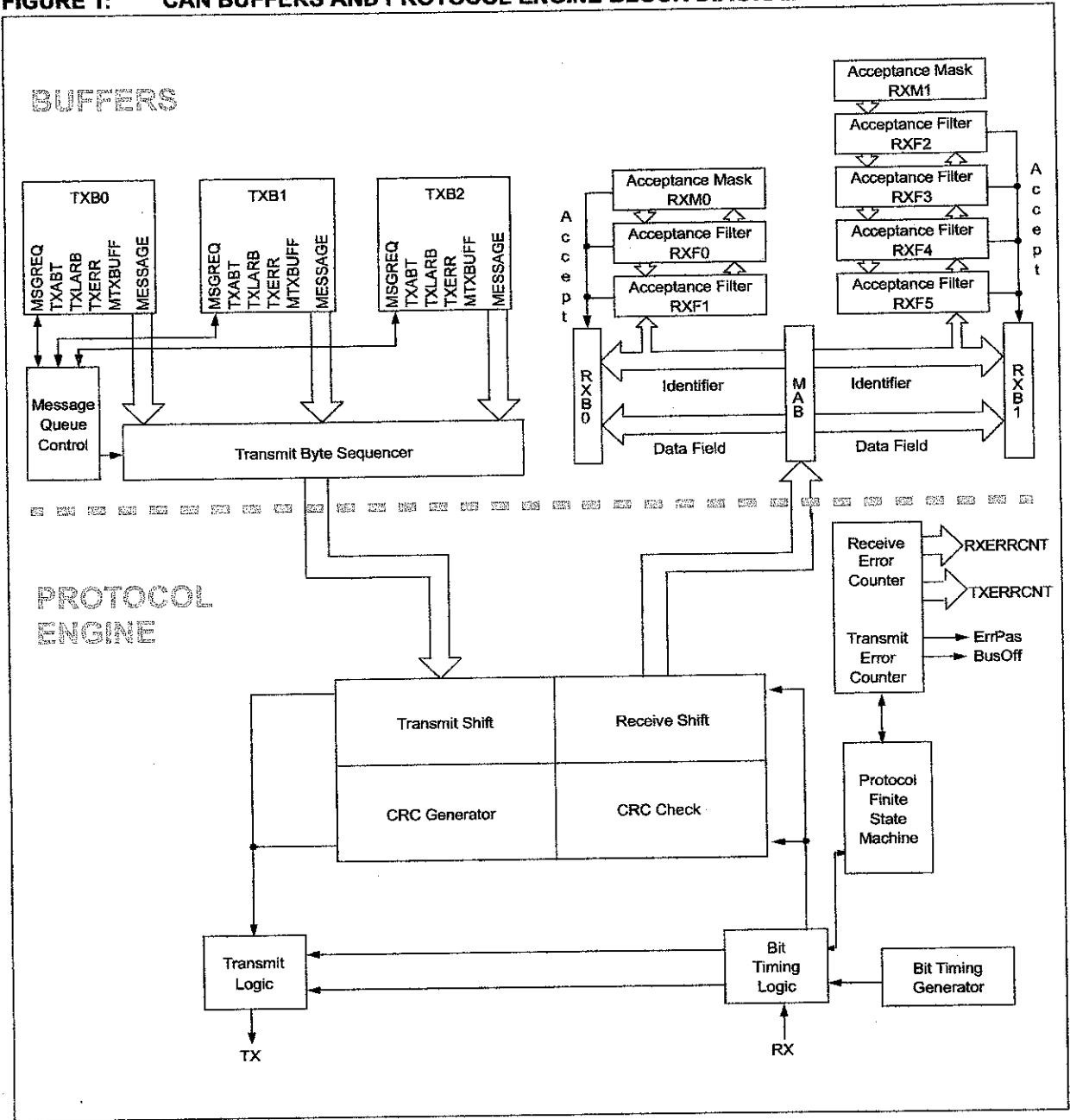
The PIC18 family of microcontrollers contain a CAN module that provides the same register and functional interface for all PIC18 microcontrollers.

The module features are as follows:

- Implementation of CAN 1.2, CAN 2.0A and CAN 2.0B protocol
- Standard and extended data frames
- 0 - 8 bytes data length
- Programmable bit rate up to 1 Mbit/sec
- Support for remote frame
- Double-buffered receiver with two prioritized received message storage buffers
- Six full (standard/extended identifier) acceptance filters: two associated with the high priority receive buffer, and four associated with the low priority receive buffer
- Two full acceptance filter masks, one each associated with the high and low priority receive buffers
- Three transmit buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low pass filter
- Programmable Loopback mode and programmable state clocking supports self test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time stamping and network synchronization
- Low Power SLEEP mode



FIGURE 1: CAN BUFFERS AND PROTOCOL ENGINE BLOCK DIAGRAM



## Bus Arbitration and Message Latency

In the CAN protocol, if two or more bus nodes start their transmission at the same time, message collision is avoided by bit-wise arbitration. Each node sends the bits of its identifier and monitors the bus level. A node that sends a recessive identifier bit, but reads back a dominant one, loses bus arbitration and switches to Receive mode. This condition occurs when the message identifier of a competing node has a lower binary value (dominant state = logic 0), which results in the competing node sending a message with a higher priority. Because of this, the bus node with the highest priority message wins arbitration, without losing time by having to repeat the message. Transmission of the lower priority message is delayed until all high priority traffic on the bus is finished, which adds some latency to the message transmission. This type of message latency cannot be avoided.

Depending on software driver implementation, additional latency can be avoided by proper design of the driver. If CAN is working at low bus utilization, then the delay in message transmission is not a concern because of arbitration. However, if CAN bus utilization is high, unwanted message latency can be reduced with good driver design.

To illustrate this point, let us examine latency that occurs because of the implementation of driver software. Consider the case when a buffer contains a low priority message in queue and a high priority message is loaded. If no action is taken, the transmission of the high priority message will be delayed until the low priority message is transmitted. A PIC18CXX8 device provides a workaround for this problem.

In PIC18CXX8 devices, it is possible to assign priority to all transmit buffers, which causes the highest priority message to be transmitted first and so on. By setting the transmit buffer priority within the driver software, this type of message latency can be avoided.

Additionally, consider the case where all buffers are occupied with a low priority message and the controller wants to transmit a high priority message. Since all buffers are full, the high priority message will be blocked until one of the low priority messages is transmitted. The low priority message will be sent only after all the high priority messages on the bus are sent. This can considerably delay the transmission of high priority messages.

How then, can this problem be solved? Adding more buffers may help, but most likely the same situation will occur. What then, is the solution? The solution is to unload the lowest priority message from the transmit buffer and save it to a software buffer, then load the transmit buffer with the higher priority message. To maintain bus control, all  $n$  Transmit buffers should be loaded with  $n$  highest priority messages. Once the transmit buffer is emptied, load the lower priority message into the transmit buffer for transmission. To do this, intelligent driver software is needed that will manage these buffers, based on the priority of the message (Lower binary value of identifier → Higher priority, see "Terminology Conventions" on page 5). This method minimizes message latency for higher priority messages.

## Macro Wrappers

One of the problems associated with assembly language programming is the mechanism used to pass parameters to a function. Before a function can be called, all parameters must be copied to a temporary memory location. This becomes quite cumbersome when passing many parameters to a generalized function. One way to facilitate parameter passing is through the use of "macro wrappers". This new concept provides a way to overcome the problems associated with passing parameters to functions.

A macro wrapper is created when a macro is used to "wrap" the assembly language function for easy access. In the following examples, macros call the same function, but the way they format the data is different. Depending on the parameters, different combinations of macro wrappers are required to fit the different applications.

Macro wrappers for assembly language functions provide a high level 'C-like' language interface to these functions, which makes passing multiple parameters quite simple. Because the macro only deals with literal values, different macro wrappers are provided to suit different calling requirements for the same functions.

For example, if a function is used that copies the data at a given address, the data and address must be supplied to the function.

### EXAMPLES

Using standard methods, a call to the assembly language function `CopyDataFunc` might look like the macro shown in Example 1.

#### EXAMPLE 1: CODE WITHOUT MACRO WRAPPER

```
#define      Address      0x1234

    UDATA
TempWord    RES    02

    banksel TempWord
    movlw   low(Address)
    movwf   TempWord
    movlw   high(Address)
    movwf   TempWord+1
    movlw   0x56      ;Copy data
    call    CopyDataFunc
```

Using a macro wrapper, the code in Example 2 shows how to access the same function that accepts the data value directly.

#### EXAMPLE 2: CODE WITH MACRO WRAPPER

```
#define      Address      0x1234

CopyData 0x56,      Address
```

The code in Example 3 shows variable data stored in `DataLoc`.

#### EXAMPLE 3: CODE WITHOUT MACRO WRAPPER

```
#define      Address      0x1234

    UDATA
TempWord    RES    02
DataLoc     RES    01

    banksel TempWord
    movlw   low(Address)
    movwf   TempWord
    movlw   high(Address)
    movwf   TempWord+1
    banksel DataLoc
    movf    DataLoc,W
    call    CopyDataFunc
```

Using a macro wrapper, the code shown in Example 4 supplies the memory address location for data instead of supplying the data value directly.

#### EXAMPLE 4: CODE WITH MACRO WRAPPER

```
#define      Address      0x1234

    UDATA
DataLoc     RES    01
CopyData_IDDataLoc,      AddressLoc
```

The code in Example 5 shows one more variation using a macro wrapper for the code of both variable arguments.

#### EXAMPLE 5: CODE WITH MACRO WRAPPER

```
    UDATA
AddressLoc   RES    02
DataLoc     RES    01

CopyData_ID_IA DataLoc,      AddressLoc
```

To summarize, the code examples previously described call for the same function, but the way they format the data is different. By using a macro wrapper, access to assembly functions is simplified, since the macro only deals with literal values.

## PIC18XXX8 CAN FUNCTIONS

All PIC18XXX8 CAN functions are grouped into the following three categories:

- Configuration/Initialization Functions
- Module Operation Functions
- Status Check Functions

The following table lists each function by category, which are described in the following sections.

**TABLE 1: FUNCTION INDEX**

Function	Category	Page Number
CANInitialize	Configuration/Initialization	6
CANSetOperationMode	Configuration/Initialization	8
CANSetOperationModeNoWait	Configuration/Initialization	9
CANSetBaudRate	Configuration/Initialization	10
CANSetReg	Configuration/Initialization	12
CANSendMessage	Module Operation	16
CANReadMessage	Module Operation	19
CANAbortAll	Module Operation	22
CANGetTxErrorCount	Status Check	23
CANGetRxErrorCount	Status Check	24
CANIsBusOff	Status Check	25
CANIsTxPassive	Status Check	26
CANIsRxPassive	Status Check	27
CANIsRxReady	Status Check	28
CANIsTxReady	Status Check	30

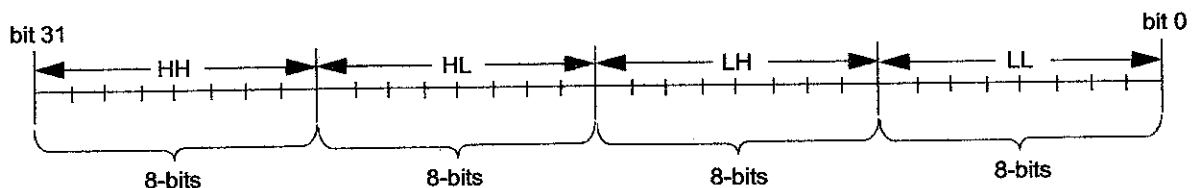
## Terminology Conventions

The following applies when referring to the terminology used in this application note.

**TABLE 2: TERMINOLOGY CONVENTIONS**

Term	Meaning
<i>xyzFunc</i>	Used for original assembly language functions.
<i>xyz</i>	The macro that will accept all literal values.
<i>xyz_I</i> (First letter of argument)	The macro that will accept the memory address location for variable implementation.
<i>xyz_D</i> (First letter of argument)	The macro that expects the user is directly copying the specified parameter at the required memory location by assembly function.

LL:LH:HL:HH



**APPENDIX D**  
**C CODE: EX\_CAN.C**

```

/////////////////////////////////////////////////////////////////
/
//////                                EX_CAN.C
//////
//////
////// Example of CCS's CAN library, using the PIC18Fxx8.  This
////// example was tested using MCP250xxx CAN Developer's Kit.
//////
//////
////// Connect PIN_B2 (CANTX) to the CANTX pin on the open NODE A of
////// the developer's kit, and connect PIN_B3 (CANRX) to the CANRX
////// pin on the open NODE A.
//////
//////
////// NODE B has an MCP250xxx which sends and responds certain canned
////// messages.  For example, hitting one of the GPX buttons on
////// the development kit causes the MCP250xxx to send a 2 byte
////// message with an ID of 0x290.  After pressing one of those
////// buttons with this firmware you should see this message
////// displayed over RS232.
//////
//////
////// NODE B also responds to certain CAN messages.  If you send
////// a request (RTR bit set) with an ID of 0x18 then NODE B will
////// respond with an 8-byte message containing certain readings.
////// This firmware sends this request every 2 seconds, which NODE B
////// responds.
//////
//////
////// If you install Microchip's CANKing software and use the
////// MCP250xxx , you can see all the CAN traffic and validate all
////// experiments.
//////
//////
////// For more documentation on the CCS CAN library, see can-18xxx8.c
//////
//////
////// This example will work with the PCM and PCH compilers.
//////

```

```

/////////////////////////////////////////////////////////////////
/
////      (C) Copyright 1996,2003 Custom Computer Services
////
//// This source code may only be used by licensed users of the CCS
////
//// C compiler. This source code may only be distributed to other
////
//// licensed users of the CCS C compiler. No other use,
////
//// reproduction or distribution is permitted without written
////
//// permission. Derivative programs created using this software
////
//// in object code form are not restricted in any way.
////
/////////////////////////////////////////////////////////////////
/

```

```

#include <18F248.h>;
#fuses HS,NOPROTECT,NOLVP,NOWDT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7
to 12

```

```

#include <can-18xxx8.c>;

```

```

int16 ms;

```

```

#int_timer2
void_isr_timer2(void) {
    ms++; //keep a running timer that increments every milli-second
}

```

```

void main() {
    struct rx_stat rxstat;
    int32 rx_id;
    int in_data[8];
    int rx_len;

```

```

//send a request (tx_rtr=1) for 8 bytes of data (tx_len=8) from id 24
(tx_id=24)
    int out_data[8];
    int32 tx_id=24;
    int1 tx_rtr=1;
    int1 tx_ext=0;
    int tx_len=8;
    int tx_pri=3;

```

```

    int i;

```

```

    for (i=0;i<8;i++) {
        out_data[i]=0;
        in_data[i]=0;
    }

```

```

    printf("\r\n\r\nCCS CAN EXAMPLE\r\n");

```

```

    setup_timer_2(T2_DIV_BY_4,79,16); //setup up timer2 to interrupt
every 1ms if using 20Mhz clock

```

```

can_init();

enable_interrupts(INT_TIMER2); //enable timer2 interrupt
enable_interrupts(GLOBAL);    //enable all interrupts (else timer2
wont happen)

printf("\r\nRunning...");

while(TRUE)
{
    if ( can_kbhit() ) //if data is waiting in buffer...
    {
        if(can_getd(rx_id, &in_data[0], rx_len, rxstat))
        { //...then get data from buffer
            printf("\r\nGOT: BUFF=%U ID=%LU LEN=%U OVF=%U ",
rxstat.buffer, rx_id, rx_len, rxstat.err_ovfl);
            printf("FILT=%U RTR=%U EXT=%U INV=%U", rxstat.filt hit,
rxstat.rtr, rxstat.ext, rxstat.inv);
            printf("\r\n    DATA = ");
            for (i=0;i<rx_len;i++) {
                printf("%X ", in_data[i]);
            }
            printf("\r\n");
        }
        else {
            printf("\r\nFAIL on GETD\r\n");
        }
    }

    //every two seconds, send new data if transmit buffer is empty
    if ( can_tbe() && (ms > 2000))
    {
        ms=0;
        i=can_putd(tx_id, out_data, tx_len, tx_pri, tx_ext, tx_rtr); //put
data on transmit buffer
        if (i != 0xFF) { //success, a transmit buffer was open
            printf("\r\nPUT %U: ID=%LU LEN=%U ", i, tx_id, tx_len);
            printf("PRI=%U EXT=%U RTR=%U\r\n    DATA = ", tx_pri, tx_ext,
tx_rtr);
            for (i=0;i<tx_len;i++) {
                printf("%X ", out_data[i]);
            }
            printf("\r\n");
        }
        else { //fail, no transmit buffer was open
            printf("\r\nFAIL on PUTD\r\n");
        }
    }
}
}

```



**APPENDIX E**  
**C CODE FOR COMMUNICATION BETWEEN TWO**  
**MICROCONTROLLERS : NODE.C**

```

//////////
//          //
// Node.c   //
//          //
//////////

#include <18F458.h>
#include <stdio.h>
#fuses XT,NOPROTECT,NOLVP,NOWDT
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7 to 12

#include <can-18xxx8.c>

int16 ms;

#int_timer2
void isr_timer2(void) {
    ms++; //keep a running timer that increments every milli-second
}

#define SEND_ID_B      0x201 //Send ID for Port B
#define GET_ID_B       0x202 //Send ID for Port C
#define RESPOND_TO_ID_B 0x203
#define RESPOND_FROM_ID_B 0x204

#define BUTTON PIN_C0
#define BUTTON_PRESSED !input(BUTTON)

void main() {

    struct rx_stat rxstat;
    int32 rx_id;
    int in_data[8];
    int rx_len;

    //send a request (tx_rtr=1) for 8 bytes of data (tx_len=8) from id 24 (tx_id=24)
    int out_data[8];
    int32 tx_id=24;
    int1 tx_rtr=1;
    int1 tx_ext=0;
    int tx_len=8;
    int tx_pri=3;

    int i;

    for (i=0;i<8;i++) {
        out_data[i]=0xFC;
        in_data[i]=0;
    }

    set_tris_c(0xFF);

    printf("\r\n\r\nCCS CAN EXAMPLE\r\n");

    setup_timer_2(T2_DIV_BY_4,79,16); //setup up timer2 to interrupt every 1ms if using 20Mhz
    clock

```

```

can_init();
can_set_mode(CAN_OP_NORMAL);

enable_interrupts(INT_TIMER2); //enable timer2 interrupt
enable_interrupts(GLOBAL);    //enable all interrupts (else timer2 wont happen)

printf("\r\nRunning...");

while(TRUE)
{
if (BUTTON_PRESSED) {
while (BUTTON_PRESSED) {}
delay_ms(200);

printf("\r\nSending message over to Node B");
can_putd(SEND_ID_B, 0, 1, 3, 1, 1);
}

if ( can_kbhit() ) //if data is waiting in buffer...
{
if(can_getd(rx_id, &in_data[0], rx_len, rxstat)) { //...then get data from buffer
if (rx_id == RESPOND_FROM_ID_B) {
printf("\r\nMessage sent to PortB");
printf("\r\nGOT: BUFF=%U ID=%LU LEN=%U OVF=%U ", rxstat.buffer, RESPOND_TO_ID_B,
rx_len, rxstat.err_ovfl);
printf("FILT=%U RTR=%U EXT=%U INV=%U", rxstat.filthit, rxstat.rtr, rxstat.ext, rxstat.inv);
//printf("\r\n  DATA = ");
for (i=0;i<rx_len;i++) {
printf("%X ", in_data[i]);
}
printf("\r\n");
}

if (rx_id == GET_ID_B) {
printf("\r\nGot a message from port B");
printf("\r\nLED ON");
output_high(PIN_D4);
can_putd(RESPOND_TO_ID_B, &i, 1,1,1,0); //put data on transmit buffer
delay_ms(2000);
output_low(PIN_D4);
}
}
}
}
}
}
}

```

**APPENDIX F**  
**C CODE FOR COMMUNICATION BETWEEN THREE**  
**MICROCONTROLLERS : STATIONA.C**

```

#include <18F458.h>
#fuses XT,NOPROTECT,NOLVP,NOWDT
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7
to 12

#include <can-18xxx8.c>

int16 ms;

#int_timer2
void_isr_timer2(void) {
    ms++; //keep a running timer that increments every milli-second
}

#define SEND_TO_PORT_B          0x201
#define SEND_TO_PORT_C          0x202
#define RESPOND_FROM_PORT_B     0x207
#define RESPOND_FROM_PORT_C     0x208

void main() {

    char out_data;
    int1 tx_rtr=1;
    int1 tx_ext=0;
    int tx_len=8;
    int tx_pri=3;
    int i;

    struct rx_stat rxstat;
    int32 rx_id;
    char in_data[8];
    int rx_len;
    int buffer=0xfc;

    printf("\r\n\r\nMonitoring CAN... Node A\r\n");

    setup_timer_2(T2_DIV_BY_4,79,16);

    can_init();
    can_set_mode(CAN_OP_NORMAL);

    enable_interrupts(INT_TIMER2);
    enable_interrupts(GLOBAL);

    printf("\r\nRunning...");

    while(TRUE)
    {

//***** Receiving Data
//*****

        if ( can_kbhit() ) //if data is waiting in buffer...
        {
            printf("\r\n\r\nData in receive buffer");
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat)) { //...then
                get data from buffer
            }
        }
    }
}

```

```

        //Respond from B
        if (rx_id == RESPOND_FROM_PORT_B) {
            printf("\r\nMessage sent to Node B");
        }
        //Respond from C
        if (rx_id == RESPOND_FROM_PORT_C) {
            printf("\r\nMessage Sent to Node C");
        }
    }
}

////////////////////////////////////
////////////////////////////////////

//*****Sending
data*****

if ( can_tbe() && (ms > 500))
{
    out_data = getchar();
    printf("\r\nTransmit buffer ready");
    printf("\r\n\n you pressed %c", out_data);
    ms=0;

    //Data for Port B
    if (out_data=='a' || out_data=='b' || out_data=='c')
    {
        i=can_putd(SEND_TO_PORT_B, out_data,
tx_len,tx_pri,tx_ext,tx_rtr); //put data on transmit buffer
        if (i != 0xFF) { //success, a transmit buffer was open
            printf("\r\nSend data to Port B");

            printf("\r\nPUT %U: ID=%LU LEN=%U ", i,
SEND_TO_PORT_B, tx_len);
            printf("PRI=%U EXT=%U RTR=%U\r\n  DATA = ", tx_pri,
tx_ext, tx_rtr);
            for (i=0;i<tx_len;i++) {
                printf("%c ",out_data);
            }
            printf("\r\n");
        }
    }

    //Data for Port C
    if (out_data=='d' || out_data=='e' || out_data=='f'){
        i=can_putd(SEND_TO_PORT_C, out_data,
tx_len,tx_pri,tx_ext,tx_rtr); //put data on transmit buffer
        if (i != 0xFF) { //success, a transmit buffer was open
            printf("\r\nSend data to Port C");
            printf("\r\nPUT %U: ID=%LU LEN=%U ", i,
SEND_TO_PORT_C, tx_len);
            printf("PRI=%U EXT=%U RTR=%U\r\n  DATA = ", tx_pri,
tx_ext, tx_rtr);
            for (i=0;i<tx_len;i++) {
                printf("%c ",out_data);
            }
            printf("\r\n");
        }
    }
}

}

////////////////////////////////////
////////////////////////////////////

```

} }

**APPENDIX G**  
**C CODE FOR COMMUNICATION BETWEEN THREE**  
**MICROCONTROLLERS : STATIONB.C**



```

#include <18F458.h>
#fuses XT,NOPROTECT,NOLVP,NOWDT
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7
to 12

#include <can-18xxx8.c>

int16 ms;

#int_timer2
void_isr_timer2(void) {
    ms++; //keep a running timer that increments every milli-second
}

#define RECEIVE_FROM_PORT_A    0x201
#define RESPOND_TO_PORT_A     0x207

void main() {

    char out_data;
    int1 tx_rtr=1;
    int1 tx_ext=0;
    int tx_len=8;
    int tx_pri=3;
    int i;

    struct rx_stat rxstat;
    int32 rx_id;
    char in_data[8];
    int rx_len;
    int buffer=0xfc;

    printf("\r\n\r\nMonitoring CAN... Node B\r\n");

    setup_timer_2(T2_DIV_BY_4,79,16);

    can_init();
    can_set_mode(CAN_OP_NORMAL);

    enable_interrupts(INT_TIMER2);
    enable_interrupts(GLOBAL);

    printf("\r\nRunning...");

    while(TRUE)
    {

//***** Receiving Data
//*****
        if ( can_kbhit() ) //if data is waiting in buffer...
        {

```

```

printf("\r\nData in receive buffer");
if(can_getd(rx_id, &in_data[0], rx_len, rxstat)) { //...then
get data from buffer

```

```

//Data from A
if (rx_id == RECEIVE_FROM_PORT_A) {
printf("\r\nData is from Node A");
printf("\r\nGOT: BUFF=%U ID=%LU LEN=%U OVF=%U ",
rxstat.buffer, rx_id, rx_len, rxstat.err_ovfl);
printf("FILT=%U RTR=%U EXT=%U INV=%U", rxstat.filthit,
rxstat.rtr, rxstat.ext, rxstat.inv);
printf("\r\n    DATA = ");
for (i=0;i<rx_len;i++) {
printf("%c ",in_data[i]);
}
printf("\r\n");
can_putd(RESPOND_TO_PORT_A, &buffer, 3, 1, 1, 0);
}
}

```

```

}
////////////////////////////////////
////////////////////////////////////

```

```

}
}

```

**APPENDIX H**  
**C CODE FOR COMMUNICATION BETWEEN THREE**  
**MICROCONTROLLERS : STATIONC.C**

```

#TYPE INT=8
#include <18F458.h>
#fuses XT,NOPROTECT,NOLVP,NOWDT
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Jumpers: 8 to 11, 7
to 12

#include <can-18xxx8.c>

int16 ms;

#int_timer2
void_isr_timer2(void) {
    ms++; //keep a running timer that increments every milli-second
}

#define RECEIVE_FROM_PORT_A      0x202
#define RESPOND_TO_PORT_A      0x208

void main() {

    char out_data;
    int1 tx_rtr=1;
    int1 tx_ext=0;
    int tx_len=8;
    int tx_pri=3;
    int i;

    struct rx_stat rxstat;
    int32 rx_id;
    char in_data[8];
    int rx_len;
    int buffer=0xfc;

    printf("\r\n\r\nMonitoring CAN... Node C\r\n");

    setup_timer_2(T2_DIV_BY_4,79,16);

    can_init();
    can_set_mode(CAN_OP_NORMAL);

    enable_interrupts(INT_TIMER2);
    enable_interrupts(GLOBAL);

    printf("\r\nRunning...");

    while(TRUE)
    {

//***** Receiving Data
*****

        if ( can_kbhit() ) //if data is waiting in buffer...
        {
            printf("\r\nData in receive buffer");
            if(can_getd(rx_id, &in_data[0], rx_len, rxstat)) { //...then

```

get data from buffer

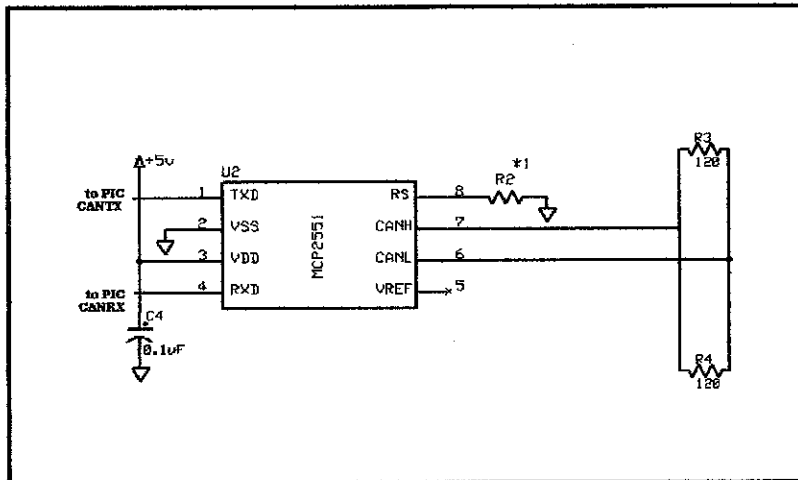
```
        //Data From A
        if (rx_id == RECEIVE_FROM_PORT_A) {
            printf("\r\nData is from Node A");
            printf("\r\nGOT: BUFF=%U ID=%LU LEN=%U OVF=%U ",
rxstat.buffer, rx_id, rx_len, rxstat.err_ovfl);
            printf("FILT=%U RTR=%U EXT=%U INV=%U", rxstat.filtHit,
rxstat.rtr, rxstat.ext, rxstat.inv);
            printf("\r\n    DATA = ");
            for (i=0;i<rx_len;i++) {
                printf("%c",in_data[i]);
            }
            printf("\r\n");
            can_putd(RESPOND_TO_PORT_A, &buffer, 3, 1, 1, 0);
        }
    }

////////////////////////////////////
////////////////////////////////////

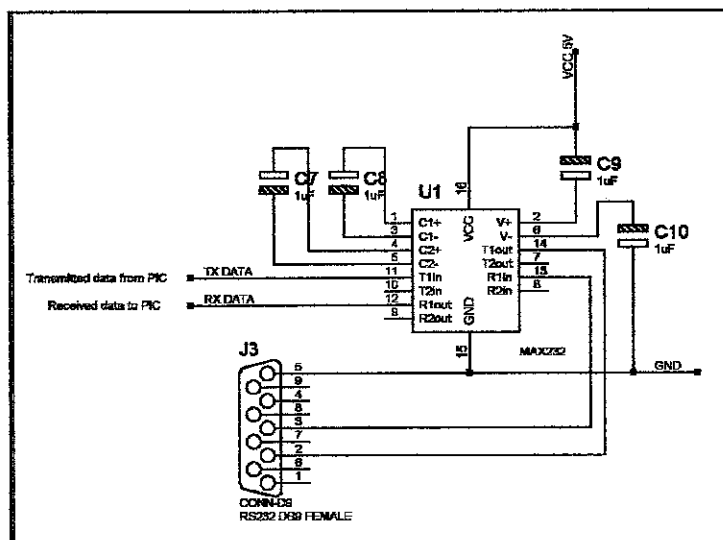
    }
}
```

## APPENDIX I

## HARDWARE CONNECTION



### CAN transceiver, MCP 2551 connection



## MAX232 hardware connection